RESEARCH ARTICLE

Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

ABDUL-MALIK H. Y. SAAD[1,2], **(Senior Member, IEEE), MOHD ZAID ABDULLAH**[3],
NAYEF ABDULWAHAB MOHAMMED ALDUAIS[4], **ANTAR SHADDAD HAMED ABDUL-QAWY**[5],
ABDULLAH B. NASSER[6], **(Member, IEEE), WAHEED ALI H. M. GHANEM**[7],
AND ADNAN HAIDER YUSEF SA'D[8]

[1]Division of Electronic and Computer Engineering, Faculty of Engineering, School of Electrical Engineering, Universiti Teknologi Malaysia (UTM), Johor Bahru, Johor 81310, Malaysia
[2]Department of Computer Engineering, Faculty of Computer Science and Engineering, Hodeidah University, Al Hudaydah, Yemen
[3]School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Nibong Tebal, Penang 14300, Malaysia
[4]Faculty of Computer Science and Information Technology (FSKTM), Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, Batu Pahat, Johor 86400, Malaysia
[5]Department of Mathematics and Computer Science, Faculty of Science, SUMAIT University, Zanzibar 1933, Tanzania
[6]School of Technology and Innovation, University of Vaasa, 65200 Vaasa, Finland
[7]Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu, Kuala Terengganu 21030, Malaysia
[8]Faculty of Computer Science and IT, Al-Razi University, Sana'a, Yemen

Corresponding authors: Abdul-Malik H. Y. Saad (eng.abdulmalik@gmail.com) and Adnan Haider Yusef Sa'd
(eng.adnanhaider@gmail.com)

ABSTRACT Fractal compression technique is a well-known technique that encodes an image by mapping the image into itself and this requires performing a massive and repetitive search. Thus, the encoding time is too long, which is the main problem of the fractal algorithm. To reduce the encoding time, several hardware implementations have been developed. However, they are generally developed for grayscale images, and using them to encode colour images leads to doubling the encoding time $3\times$ at least. Therefore, in this paper, new high-speed hardware architecture is proposed for encoding RGB images in a short time. Unlike the conventional approach of encoding the colour components similarly and individually as a grayscale image, the proposed method encodes two of the colour components by mapping them directly to the most correlated component with a searchless encoding scheme, while the third component is encoded with a search-based scheme. This results in reducing the encoding time and also in increasing the compression rate. The parallel and deep-pipelining approaches have been utilized to improve the processing time significantly. Furthermore, to reduce the memory access to the half, the image is partitioned in such a way that half of the matching operations utilize the same data fetched for processing the other half of the matching operations. Consequently, the proposed architecture can encode a 1024×1024 RGB image within a minimal time of 12.2 ms, and a compression ratio of 46.5. Accordingly, the proposed architecture is further superior to the state-of-the-art architectures.

INDEX TERMS Digital circuits, field programmable gate arrays, fractal colour image compression, parallel architectures, pipeline processing.

I. INTRODUCTION

In recent years, images are found everywhere and become an essential element in our daily lives. Images are produced

The associate editor coordinating the review of this manuscript and approving it for publication was Yun Zhang.

intensively every single day in various forms such as medical images, personal images, social media images, surveillance images, and graphics images. Storing these images or transferring them through the network creates an unbearable burden for memory devices and the network bandwidth too. Taking this into account, compression is an indispensable tool

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE *Access*

for archiving images in fewer spaces of storage and also for transferring images in less time of transmission.

Fractal image compression (FIC) is one of the most popular techniques due to its distinctive way in compressing images. This compression technique uses self-similarity features as a means to compress an image [5]. It also offers a high compression ratio (CR), especially when applied to a digital image with a high degree of self-similarity like aerial photography or satellite imagery [7]. Owing to its popularity in digital archiving, FIC has been found in numerous applications such as character recognition [8], super-resolution [9], watermarking [10], [11], and digital signature embedding [12]. FIC also possesses other attractive features like good peak signal-to-noise ratio (PSNR) performance and a simple decoding method [13], [14], [15], [16], [17], [18]. However, FIC suffers one major drawback arising from the computational complexity of the algorithm. Typically, FIC requires a very large number of searches to find excellent or good-enough maps between image blocks. Theoretically, the time complexity of the fractal compression algorithm approximately equals to $O\left(n^4\right)$ for $n \times n$ size image [19], [20]. Owing to this, the encoding time required for FIC is generally large and not efficient for most real-time applications that require the processing of 30 frames per second, or more.

To date, various methods have been proposed for accelerating the fractal algorithm. These methods can be grouped into two types: software- and hardware-based methods. The software-based methods mainly focus on reducing the size of the range and domain pool, adopting of classification approach [21], restriction of search space [22], [23], combining of different code schemes [24], and applying of feature vector approach [25]. For instance, Bani-Eqbal [26] arranged the domain blocks in a tree structure to reduce the number of candidate blocks for matching search. In an earlier study, Saupe and Hamzaoui [27] discarded the domain blocks that have low variance. Tong and Pi [28] presented an adaptive search for excluding the domain blocks that do not satisfy the necessary condition of better matching. Meanwhile, Tong and Wong [29] converted the matching search problem to a nearest neighbour search problem. Although these schemes are successful in speeding up the FIC, the attained encoding times are still too high and not adequate to deliver the real-time requirement for the most time-sensitive applications. So far, this problem is still existing even when using cutting-edge microprocessors and digital signal processors (DSPs). This is because the processors architectures are still sharing the same concept as the older processor, where a static set of instructions is run in a static architecture. On the other hand, the hardware-based methods execute FIC operations in parallel, resulting in the enhancement of the encoding time performance.

Some research works targeting hardware implementation of FIC have been published recently [1], [2], [30], [31], [32], [33]. These works exploit the inherent parallelism in FIC as means of speed-up. Although these works reduce the encoding time significantly, they are developed to encode grayscale images only. However, most of the images existing these days are colour images. This shortage of fractal-colour-image-compression (FCIC) hardware implementation can be attributed to the fact that designing a dedicated hardware to process colour images is much more complicated compared to designing a custom hardware for processing grayscale images. Due to this, fractal coding designers were contented with the straight-forward method (i.e., three-component Separated Fractal Coding (SFC)) which encodes colour images using the grayscale-based methods. In this case, each colour component in a colour image is treated as a single grayscale image. As a result, the correlation among colour components is not exploited which results in a relatively low compression ratio and PSNR. Furthermore, the encoding time is increased significantly.

To the best of our knowledge, there is no certain hardware work developed for encoding colour images except that proposed in [34]. According to [34], a soft-core processor designed for Xilinx FPGA has been used for coding a 64×64 colour image. In the process, the RGB components of the colour image are transformed into YUV components, and then fractal coding is performed in different sampling modes of 4:4:4, 4:2:2, 4:2:0, and 4:1:1 for attaining a higher compression rate. The range and domain blocks for each component are classified into a specific number of classes. More specifically, Y's blocks are classified into 72 classes while U's and V's blocks are classified into 24 classes each. The encoding process is performed so that each component is encoded separately without exploiting the inter-colour-component redundancy. Although the encoding time of this implementation is not reported, the time can be estimated by considering an almost similar implementation published by the same authors Son, et al. [35], but applied for the grayscale images. Based on this assumption, the encoding time of colour image is approximately 33 s. This reported time is large because this design lacks a hardware accelerator unit and also because it still encodes colour components individually.

In this paper, a new high-speed pipelined parallelism architecture is proposed for coding a high-resolution colour image of 1024×1024 pixels size. Since Field-Programmable-Gate-Arrays (FPGAs) have the advantage to feature dense fine parallelism where hundreds of instructions can be executed simultaneously, as well as be reprogrammable, therefore, the proposed architecture is realised and developed on FPGA. The proposed architecture is designed to exploit the cross-correlation between colour components in order to achieve higher CR and lower encoding time. Unlike YUV and other colour spaces, the colour components in RGB space are relatively more correlated and have a higher degree of similarity [36]. Therefore, colour images are chosen to be encoded in RGB space, and thus direct mapping between the colour components is used to reduce the search process. By doing so, one component is encoded with a search-based method, while the other two components are encoded with a searchless-based method. As a result of encoding two of the three colour

IEEE *Access*

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

components using the searchless method, the runtime is nearly one-third of that when using grayscale-based designs. Furthermore, it provides relatively higher compression rate and image quality compared to the conventional approach that encodes colour components separately. The main two contributions of this work are:

- Develop a hardware-friendly fractal RGB image encoding method with two encoding schemes, utilizing the inter-correlation between the colour components in RGB images to improve the encoding speed and the compression rate.
- Design a high-speed hardware architecture of fractal-based colour image compression with deep pipeline processing and two parallel matching processors.

II. MATHEMATICAL BACKGROUND OF FRACTAL ALGORITHM

In this section, the mathematical background of the fractal algorithm is demonstrated and explained in detail. The encoder and decoder parts of the fractal algorithm are elaborated clearly in the following sub-sections.

A. ENCODING PROCEDURE

The idea of the fractal algorithm is to find a set of transformations that can map an image into itself. In other words, the fractal algorithm attempts to find the local self-similarities on the image and maps them together using affine transformation. Thus, to encode an image, the image needs to be partitioned into a set of non-overlapping blocks referred to as range blocks, Rs. Additionally, the image is also partitioned into larger blocks called domain blocks and denoted as Ds. Unlike the range blocks, the domain blocks can be overlapped together and are generally $4\times$ greater than the range blocks. In the case that the range blocks have the size of $\alpha \times \alpha$ pixels, the size of the domain blocks will be generally $2\alpha \times 2\alpha$.

In order to map the range and domain blocks together, the encoder will search the domain pool for the acceptable or the best matched-domain block for each range block on the image. As said earlier, the affine transformation needs to be applied to the candidate domain blocks in order to maximize the matching degree between the matched blocks. Thus, the values of the transformer's coefficients for the best matches are then stored as the fractal codes for the encoded image. Typically, the affine transformer can be simplified to three small basic functions. First, the scaling function is responsible to contract the domain blocks to the size of the range block. To do so, every 2×2 pixels in the respective domain block is replaced by one pixel of their average value. Second, the geometric function where the domain block can be rotated and reflected by specified degrees to generate several symmetries of the domain block. Last is the massic function where the intensity values of the candidate domain are adjusted by two parameters; contrast scaling δ and brightness offset ρ parameters. The scaling parameter is typically between -1 and 1 [37].

B. DECODING PROCEDURE

In order to decode an image being encoded with the fractal algorithm, the decoder requires to perform an iterative reconstructing process. The first task of the decoder is to use an arbitrary image and then update the image's partitions as follows:

$$R = \delta \times D_j + \rho \times I \qquad (1)$$

where R represents the retrieved range block and D_j represents the mapped j^{th} domain block. The δ, ρ and j coefficients are the stored fractal codes for the corresponding range block. Once it finishes retrieving all the range blocks on the image, the produced image will be used as the domain image for the next iteration. This process will continue till the encoded image is retrieved. Typically, 10 to 20 iterations are enough to reconstruct an image encoded by the fixed domain block size approach [38]. To measure the image quality for the reconstructed RGB images, the PSNR metric is usually used and computed as follows:

$$PSNR = 10 \log_{10} \frac{255^2}{\frac{1}{M \times N \times 3} \sum (\nu - \tilde{\nu})^2} \qquad (2)$$

where ν and $\tilde{\nu}$ are, respectively, the original and retrieved RGB images, and $M \times N$ is the height $\times$ width of the encoded image, which is here 1024×1024.

III. METHODOLOGY
A. PROPOSED FCIC ALGORITHM

In the proposed algorithm, the colour components of an RGB image are processed with two different schemes, i.e., search and searchless-based schemes. The search scheme is used to encode the most-correlated component, while the searchless scheme is used for encoding the remaining two components. From [36], it can be found that R and B components are highly correlated to the G component and consequently the most-correlated component is G. Therefore, G-component is denoted here as the primary component while R and B components are denoted as the secondary components. In this case, the primary component is processed with the search scheme and the secondary components are processed with the searchless scheme. The procedures for both encoding schemes are discussed in the following sub-sections.

1) SEARCH SCHEME

As previously discussed in Section 2.1, the kernel of the fractal method is to pair each range block with an appropriated domain block. Thus, each range block in the G-component; R^g; needs to be mapped to one domain block, D^g. The selected domain block must have the minimum matching error compared to others. In fact, searching the entire component for the most matched block is a very time-consuming operation as it requires performing a massive number of matching operations. To reduce the required number of matching operations, the searched blocks are restricted to those that exist only in the area close to the corresponding
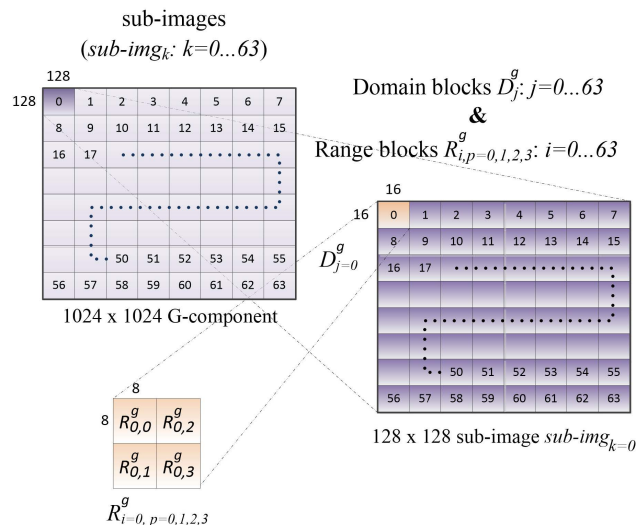
A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE *Access*

FIGURE 1. Steps in partitioning G component.

range block. This generally will not cause noticeable degradation in the quality of the decoded image as the correlation among the adjacent blocks is generally high, specifically for images captured from natural scenes [5], [39], [40]. Therefore, in this work, each range block R^g in a 128×128 window is compared with every domain block D^g in the same window. With this selected window size, 1024×1024 G-component would contain 64 non-overlapped windows. In this case, G-component is firstly divided into non-overlapping sub-images (denoted as *sub-img*) of 128×128 pixels size each. This will result in 64 $sub\text{-}img_k : k = 0 \cdots 63$. Then, each sub-image is partitioned into range and domain blocks of 8×8 and 16×16 pixels size each, respectively. As a result, there will be 64 non-overlapped domain blocks in each sub-image, and they are denoted as $D^g_j, j = 0 \cdots 63$. These 64 domain blocks constitute one search domain pool. In terms of the range block, there are 256 blocks in each sub-image and these blocks are grouped into 64 sets of four-adjacent range blocks denoted as $R^g_{i,p=0,1,2,3} : i = 0 \cdots 63$. As a result, every four adjacent range blocks $R^g_{i,p=0,1,2,3}$ will constitute a domain block D^g_j, where $j = i$. Fig. 1 illustrates the overall partitioning procedures for G-component.

To match $R^g_{i,p}$ with D^g_j, the affine transform is used to maximize the matching degree between the matched blocks. For a simple implementation and fewer matching operations, the geometric function involved in the affine transform is discarded. In this case, the affine transform ψ for D^g_j is computed as:

$$\psi\left(D^g_j\right) = \delta \times \gamma(D^g_j) + \rho \times I \qquad (3)$$

where γ is the contract function returning the contracted domain block of the size $\alpha \times \alpha$, I is the identity matrix of the size $\alpha \times \alpha$ where all entries are ones, and δ and ρ are, respectively, the scaling and offset parameters.

Now, to measure the similarity between $R^g_{i,p}$ and the obtained transform of D^g_j (i.e., $\psi\left(D^g_j\right)$), the sum-of-absolute-differences (SAD) metric is used. Among the common metrics (e.g., the least-squared-error (LSE) and the mean-squared-error (MSE)), SAD is more hardware-friendly as it does not require multiplication operations. The formula for computing SAD is as follows:

$$SAD\left(R^g_{i,p}, \psi\left(D^g_j\right)\right) = \frac{1}{N}\sum_{x=1}^{N}\left|R^g_{i,p,x} - \psi\left(D^g_j\right)_x\right| \qquad (4)$$

where $R^g_{i,p,x}$ and $\psi\left(D^g_j\right)_x$ are, respectively, the x_{th} intensity values of the respected range and transformed domain blocks, and N is the number of pixels in $R^g_{i,p}$; i.e., $N = 8 \times 8 = 64$. For getting less distortion and max matching degree, ρ is computed as:

$$\rho = \mu\left(R^g_{i,p}\right) - \delta \times \mu(D^g_j) \qquad (5)$$

where $\mu\left(R^g_{i,p}\right)$ is the mean value for $R^g_{i,p}$, $\mu(D^g_j)$ is the mean value for D^g_j and δ is the contrast scaling parameter. To gain a high compression rate, δ is chosen to be a 2-bit size and, therefore, it has four possible values (i.e., $-0.5, 0.25, 0.5, 1$). In this case, each of these values must be tested in order to select the one that produces less distortion.

For a given range block $R^g_{i,p}$, the encoder will search the domain pool $D^g_j, j = 0, \ldots, 63$ for the best-matched domain block that produces the minimum SAD value. The corresponding δ and ρ values which represent the fractal codes require to be stored together with the spatial information of the matched domain block. In this work, ρ and δ are of 7-bit and 2-bit sizes, respectively.

From Fig. 1, it is clear that each domain block contains four range blocks. Thus, reading four adjacent range blocks $R^g_{i,p=0,1,2,3}$ from memory will equal reading a domain block. Likewise, reading one domain block will implicitly lead to reading four range blocks. Therefore, when a domain block is read from the memory for similarity check, the implicit four range blocks can be utilised and processed at the same time, resulting in reduced memory access. Consequently, the compression time can be reduced effectively.

In the process, each four adjacent $R^g_{i,p=0,1,2,3}$, $i = 0 \cdots 63$ is compared consecutively with all $D^g_j, j = 0 \ldots 63$ existing in the same sub-image. At each comparison between $R^g_{i,p=0,1,2,3}$ and D^g_j, the implicit blocks (i.e., $D^g_{j=i}$ and $R^g_{i=j,p=0,1,2,3}$, respectively) are utilised to perform another matching operation. Thus both matchings $match(R^g_{i,p}, D^g_j)$ and $match(R^g_{i=j,p}, D^g_{j=i})$ can be executed in parallel using two matching processors and the results for the best-matched blocks need to be stored. To avoid performing the same matching twice, the $match(R^g_{i,p}, D^g_j)$ is carried out in one processor for $j \geq i$ while the $match(R^g_{i=j,p}, D^g_{j=i})$ for $j < i$ is carried out in other one. For example, for $i = 0$, the four range blocks $R^g_{0,p=0,1,2,3}$ will be matched one-by-one with $D^g_j : j = 0 \cdots 63$ in the first processor, while the other processor will

IEEE *Access*

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

carry on the matching between $R_{i=j,p}^g : j = 1 \cdots 63$ and $D_{j=0}^g$. Next, for $i = 1$, the next 4-range block $R_{i=1,p=0,1,2,3}^g$ will require to be compared with $D_j^g : j = 1 \cdots 63$ only because the range blocks $R_{i=1,p=0,1,2,3}^g$ have already been compared with $D_{j=0}^g$ in the first cycle. Meanwhile, the matching between $R_{i=j,p}^g : j = 2 \cdots 63$ and $D_{j=1}^g$ will be carried out on the second processor. Therefore, for each upcoming value of i, the number of domain blocks that need to be matched is $64 - i$ as $j = i, \ldots, 63$. As a result, the number of matching operations is reduced when the value of i is increased. This leads to a significant reduction in search time.

2) SEARCHLESS SCHEME

Within this scheme, the high degree of similarity between the colour components is utilised to encode R and B components in a significantly short time. Since the range blocks from G component, R^g, are strongly correlated with the range blocks from R and B components (R^r and R^b) when they are totally overlapped, therefore, both R^r and R^b are mapped with its overlapped R^g. For doing so, R and B components need to be partitioned in a similar way that the G-component is partitioned. One exception is that, unlike G-component that needs to be partitioned into range and domain blocks for pairing each range block with a particular domain block, R and B components need to be partitioned only into range blocks as they will be paired to the range blocks in G-component. In this case, R and B components are partitioned into 64 sub-images each and each of them is partitioned into 8×8 non-overlapping sets of four range blocks called $R_{i,p=0,1,2,3}^r$ and $R_{i,p=01,2,3}^b$, respectively, where $i = 0, \ldots, 63$. For each range block in the secondary components, the affine transform of the mapped block, say $R_{i,p}^g$, is computed as follows:

$$\psi\left(R_{i,p}^g\right) = \delta \times R_{i,p}^g + \rho \times I \tag{6}$$

where here the contract function is removed as $R_{i,p}^g$ has the same size as $R_{i,p}^r$ and $R_{i,p}^b$. The two parameters, δ and ρ, are defined in the same fashion as described in Section 3.1.1. Thus, the values of these parameters are stored as the fractal codes for each $R_{i,p}^r$ and $R_{i,p}^b$. The flowchart of the proposed encoding method for RGB images is shown in Fig. 2.

In respect to the decoding process, the R and B components are decoded in a different way compared to the G-component. In fact, the G-component is decoded in a conventional manner as described in Section 2.2. Unlike the G component which is reconstructed from an arbitrary image with numerous iterations, the R and B components are reconstructed from the retrieved G component and only need one iteration to be reconstructed. Consequently, the decoding time is effectively decreased, and this is another advantage of the introduced encoding approach.

B. FCIC ARCHITECTURE

To achieve a high-speed encoding using the proposed FCIC algorithm, new hardware architecture is proposed as shown
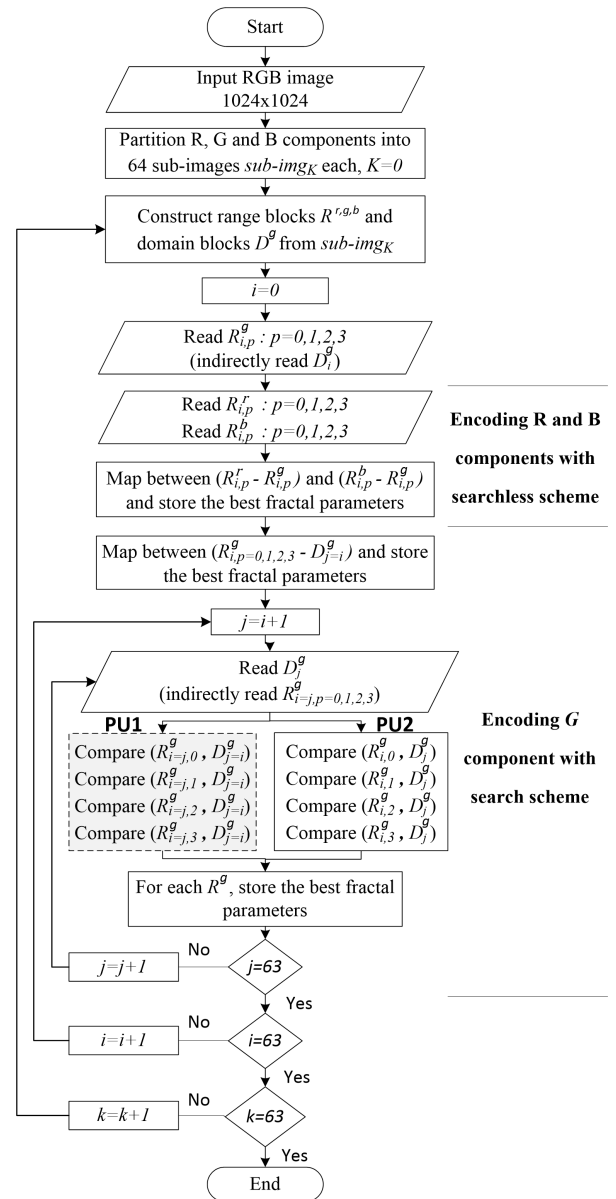

FIGURE 2. Flowchart of the proposed FCIC algorithm.

in Fig. 3. The proposed architecture is designed carefully to work at a higher clock speed and to provide larger throughputs. To do so, each component of the proposed architecture is designed and developed to have lower latency. Parallelism and pipelining approaches have been used widely and efficiently in designing the whole system in order to reach optimal performance. Basically, the proposed architecture comprises (i) Memory-Control (MemCtrl) unit, (ii) Memory-Addresses-Generator (MemAddrG) unit, (iii) Mean-and-Contraction Computing (MCC) unit, (iv) two matching processors (PU1 and PU2), and several control units. Examples of the developed control units are (i) the Storing-Control-unit (SCtrl), (ii) MCC-Ctrl unit and (iii) the Fractal Codes and SAD Storing-Control-unit (FC-SAD SCtrl). Each matching processor in the system contains RAMs, registers,
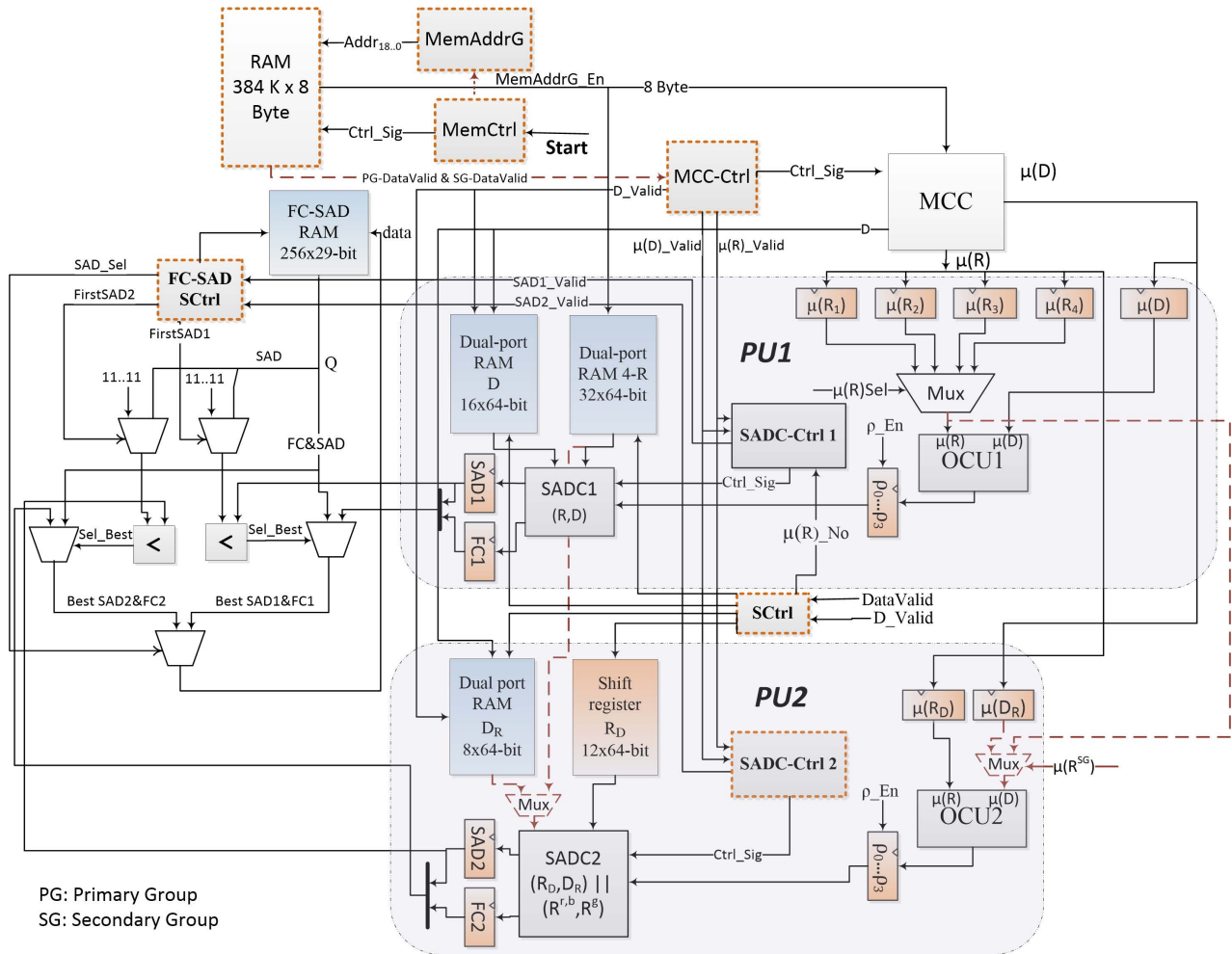
A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE Access

FIGURE 3. Overall hardware architecture for encoding colour RGB images with fractal compression technique.

an Offset-Computation-Unit (OCU), and a SAD-Computation (SADC) unit with its control unit (SADC-Ctrl).

In operation, the MemAddrG unit is designed to generate the required addresses in order to fetch the image blocks as described in Section 3.1., while the MemCtrl unit is designed to control the main memory. The MCC unit is used to compute the mean values for the fetched range and domain blocks. Besides, it is responsible to contract the domain blocks' size to 8×8 pixels. The computation of the offset parameter and the sum of absolute differences for each mapping is performed on OCU and SADC units, respectively. These units are controlled by MCC-Ctrl and SADC-Ctrl modules, respectively. Last but not least, SCtrl unit is to control the storing of the fetched range and domain blocks in the internal memory for computing SAD values, while FC-SAD SCtrl is to store the best fractal codes. For a clear understanding of the encoding process, the memory organisation, and the memory-addresses-generator unit MemAddrG are demonstrated first.

1) MEMORY ORGANISATION
A 1024×1024 RGB image of 24-bit pixel size requires 3 megabytes of memory in order to be fully stored in the system. With a memory word size of 64-bit, the encoded image requires 3×2^{17} memory words or locations. The image's pixels are buffered in a row-wise order, where the colours components are buffered one after another. Thus, the address bus width is 19-bit. As shown earlier that the encoded image comprises three components, where each component is subdivided into 64 sub-images and each is further divided into 64 domain and/or 256 range blocks. To read a particular $R_{i,p}$ or D_j block from a certain sub-image $sub\text{-}img_k$ in one of the colour components, altogether the block, the sub-image and the component spatial information (i.e., i, j and k) are required to generate the memory addresses. In this case, two bits are required to address the colour components, six bits to address the sub-images, six bits to address the domain/4-adjacent-range blocks, and five bits to address the pixels in the respective image block. Thus, the 19-bit long address $Addr_{18...0}$ is constructed as: $Addr_{18,17}$ is constructed from the colour component index (i.e., 00, 01 and 10 for G, R, and B components, respectively), $Addr_{16...14}$ and $Addr_{13...11}$ are constructed from the three most-significant bits of the sub-image and domain block indices respectively (i.e., k and j), $Addr_{6...4}$ and $Addr_{3...1}$ are constructed from the
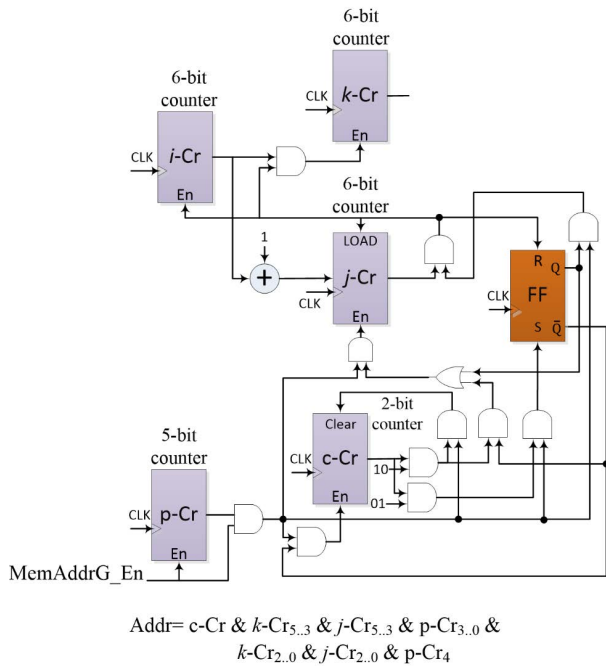
IEEE Access

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

FIGURE 4. MemAddrG unit.

Addr= c-Cr & k-Cr$_{5..3}$ & j-Cr$_{5..3}$ & p-Cr$_{3..0}$ &
k-Cr$_{2..0}$ & j-Cr$_{2..0}$ & p-Cr$_4$

remaining three least-significant bits of the sub-image and domain block indices respectively, $Addr_{10...7}$ and $Addr_0$ are to read the domain/4-adjacent-range block's pixels. Correspondingly, the base addresses of each sub-image and image block are in the form of "*xx-xxx-000-0000-xxx-000-0*" and "*xx-xxx-xxx-0000-xxx-xxx-0*".

2) MEMORY-ADDRESSES-GENERATOR UNIT

To generate the required addresses for fetching the image blocks as described earlier, we designed a MemAddrG unit as shown in Fig. 4. From this figure, it is clear that the unit is equipped with five counters, a flip-flop, an adder, and several logic gates. The counters are component counter *c-Cr*, sub-image counter *k-Cr*, 4-range block counter *i-Cr*, block counter *j-Cr*, and block's pixels counter *p-Cr*. All counters are of 6-bit size except *c-Cr* and *p-Cr* which are 2-bit and 5-bit sizes, respectively.

Referring to Fig. 4, the *p-Cr* is enabled whenever the signal *MemAddrG_En* is in HIGH state. This leads to generating the addresses required for reading all pixels in the desired domain or 4-range blocks. Since these blocks are of 16×16 pixels size and stored in 32 memory words of 8-byte, the addresses for the left 16×8 pixels are generated first, followed by the addresses of the right 16×8 pixels. As a result, the range blocks are read in the right sequence, i.e., $R_{i,0}$, $R_{i,1}$, $R_{i,2}$ and $R_{i,3}$. The counters *i-Cr*, *j-Cr*, and *k-Cr* are responsible to provide the right indices for the desired read block based on the flowchart shown in Fig. 2, while the counter *c-Cr* is used to select the colour component to be read (i.e., 00, 01 and 10 for reading G, R and B components, respectively).

For encoding the image as described in 3.1, the MemAddrG unit first generates the 32 addresses for reading $R_{0,p=0,1,2,3}^{G}$ by enabling the counter *p-Cr*, resulting in counting from "00000" ... "11111". Meanwhile, the remaining counters are all having zeros values. Once *p-Cr* reaches the final count (*p-Cr* = "11111"), *c-Cr* is enabled to allow generating the base addresses of the overlapped range blocks on R and B components. Consequently, the first four range blocks of all colour components are read in the following sub-sequence: $R_{0,p=0,1,2,3}^{g}$, $R_{0,p=0,1,2,3}^{r}$, $R_{0,p=0,1,2,3}^{b}$. In this case, both $R_{0,p=0,1,2,3}^{r}$ and $R_{0,p=0,1,2,3}^{b}$ can be encoded with the proposed searchless scheme by mapping them to $R_{0,p=0,1,2,3}^{g}$.

At the end of generating the required addresses for reading $R_{0,p=0,12,3}^{b}$, the values of the counters *p-Cr* and *c-Cr* are equal to "11111" and "10", respectively. For the next cycle, *c-Cr* is cleared, and *j-Cr* is enabled in order to generate the addresses for reading the next block at the G component, i.e., D_1^g. This allows comparing $R_{0,p=0,1,2,3}^{g}$ with D_1^g. Since R_0^g needs also to be compared with another 62 domain blocks (i.e., D_1^g to D_{63}^g), *j-Cr* is enabled for one clock cycle at the end of fetching each domain block, resulting in to increment *j*. To prevent *c-Cr* be enabled at the same time, a flip-flop (FF) is used. Accordingly, *c-Cr* can be only enabled when the FF output, *Q*, is zero, while *j-Cr* can be enabled when *Q* is one. In this case, the FF is set when the unit starts generating R_0^b addresses and reset when it completes generating the addresses of the last domain block D_{63}^g (*j-Cr*= 63).

Once the values of the counters *j-Cr* and *p-Cr* reach the maximum value, *j-Cr* is loaded with *i-Cr* + 1, which leads to fetching the subsequent 4-range block, $R_{1,p=0,1,2,3}^{g}$. At the same time, the counter *i-Cr* is also enabled. When the counters altogether *i-Cr*, *j-Cr* and *p-Cr* reach their final counts, this means that the current sub-image is completely read and the MemAddrG unit needs to start generating the addresses for fetching the next sub-image, *sub-img*$_{k=1}$. Thus, the counter *k-Cr* is enabled for one clock cycle to generate the base address of the next sub-image. Then the counters c-Cr, *i-Cr*, *j-Cr* and *p-Cr* are enabled in the same manner as for the previous sub-image. This process continues until all sub-images are read and encoded.

3) ENCODING PROCESS DATAFLOW

To let the system starts the encoding process, the signal *Start* needs to be asserted. At this time, the MemCtrl unit enables the MemAddrG unit by asserting the control signal, *MemAddrG_En*. This control signal *MemAddrG_En* and some others such as *PG-DataReady, SG-DataReady* are also provided by MemCtrl unit. *PG-DataReady* and *SG-DataReady* control signals are asserted to indicate that the pixels values to be read from the primary component or from the secondary components, respectively, are available on the data bus. Thus, they give an indication to other units to start processing the data with the corresponding encoding scheme.

The implementation of each encoding scheme requires fetching different blocks as described previously. According

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE *Access*

to MemAddrG, in every cycle, four range blocks of every colour component (R_i^g, R_i^r, R_i^b) are read first, followed by reading the domain blocks from the G component. Since R_i^r and R_i^b need only R_i^g to be encoded with the proposed searchless scheme, therefore, the searchless scheme is carried out first. Then, the proposed search scheme is executed for encoding R_i^G.

As stated previously, R_i^g, R_i^r and R_i^b are consecutively fetched for encoding R_i^r and R_i^b with the searchless scheme. At the time of reading $R_{i,p=0,1,2,3}^g$, MCC computes the mean value of each range block. Simultaneously, StrCtrl stores $R_{i,p=0,1,2,3}^g$ and their computed mean values in the dual-port RAM 4-R and $\mu\left(R_{0,1,2,3}\right)$ registers, respectively. As shown in Fig. 3, these memory elements are available in the processor *PU1*. On the other hand, R_i^r and R_i^b range blocks and their mean values are stored in the internal memory of PU2, which are respectively 64 12-bit shift register and $\mu\left(R_D\right)$ register. For each range block of R_i^r and R_i^b, the matching process is performed once its mean value is available in $\mu\left(R_D\right)$. As MCC is designed to compute the mean value with 3 pipelining stages as shown in Fig. 5, the mean value needs three clock cycles to be computed and one additional cycle to be stored in the respective register $\mu\left(R_D\right)$. Thus, after four clock cycles of reading R_i^r or R_i^b, the processor *PU2* starts the matching process by computing the offset value g. When computing ρ using OCU in *PU2*, the two multiplexers, *mux1* and *mux2*, select the corresponding $\mu\left(R_{i,p}^g\right)$ value to be fed to the OCU $\mu(D)$ input. For each scaling value δ, OCU produces one value of ρ which is stored in the corresponding register at the 5th clock cycle. At this stage, the PU2 starts computing the SAD value for each set of δ and ρ using its SADC unit. To do so, the corresponding $R_{i,p}^g$ block needs to be provided to SADC unit. Since $R_{i,p}^g$ is currently available in the internal dual-port RAM 4-R, therefore, $R_{i,p}^g$ is provided to SADC unit from this internal RAM. On the other hand, the 12-bit shift register would provide the processed range block (i.e., $R_{i,p}^r$ or $R_{i,p}^b$) to SADC at the required time. For each set of δ and ρ, SADC computes the SAD value, and these computed values are then compared together to select the minimum SAD, *MinSAD*. The SADC unit is designed to do the given task in 4-pipeline stages. The encoding set corresponding to the minimum SAD is stored for decoding purpose. Thus, these operations are repeated for each range block of R_i^r and R_i^b in the secondary components. Fig. 6 demonstrates the pipeline processing for encoding the range blocks of the secondary components ($R_{i,p}^r$ and $R_{i,p}^b$) by the proposed architecture.

After reading the range blocks R_i^r and R_i^b for encoding them by the searchless scheme as shown in Fig. 6, MemCtrl starts immediately reading the domain blocks from the primary component in order to encode R_i^g with the search scheme. Unlike R_i^r and R_i^b, R_i^g is encoded using both PU1 and PU2 processors. In this case, *PU1* executes the matching operations between each four-range block $R_{i,p=0,1,2,3}^g$ and domain blocks $D_j^g : j = i \cdots 63$. On the other hand, *PU2* executes the matching operations between each domain block $D_{j=i}^g$ with
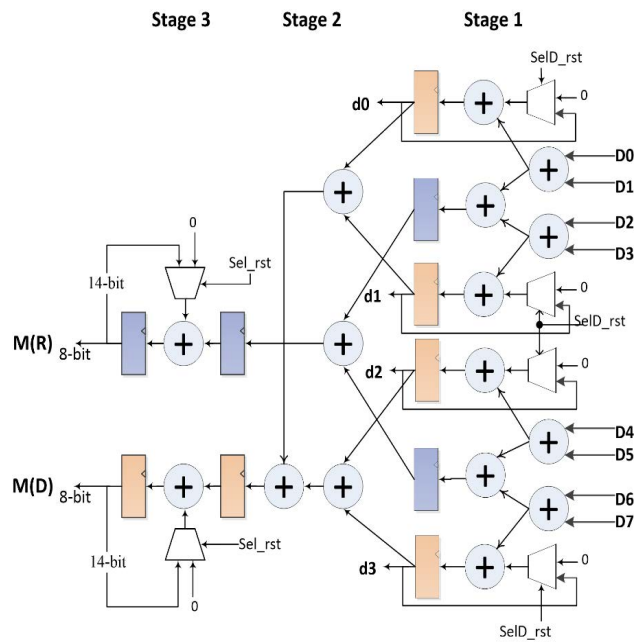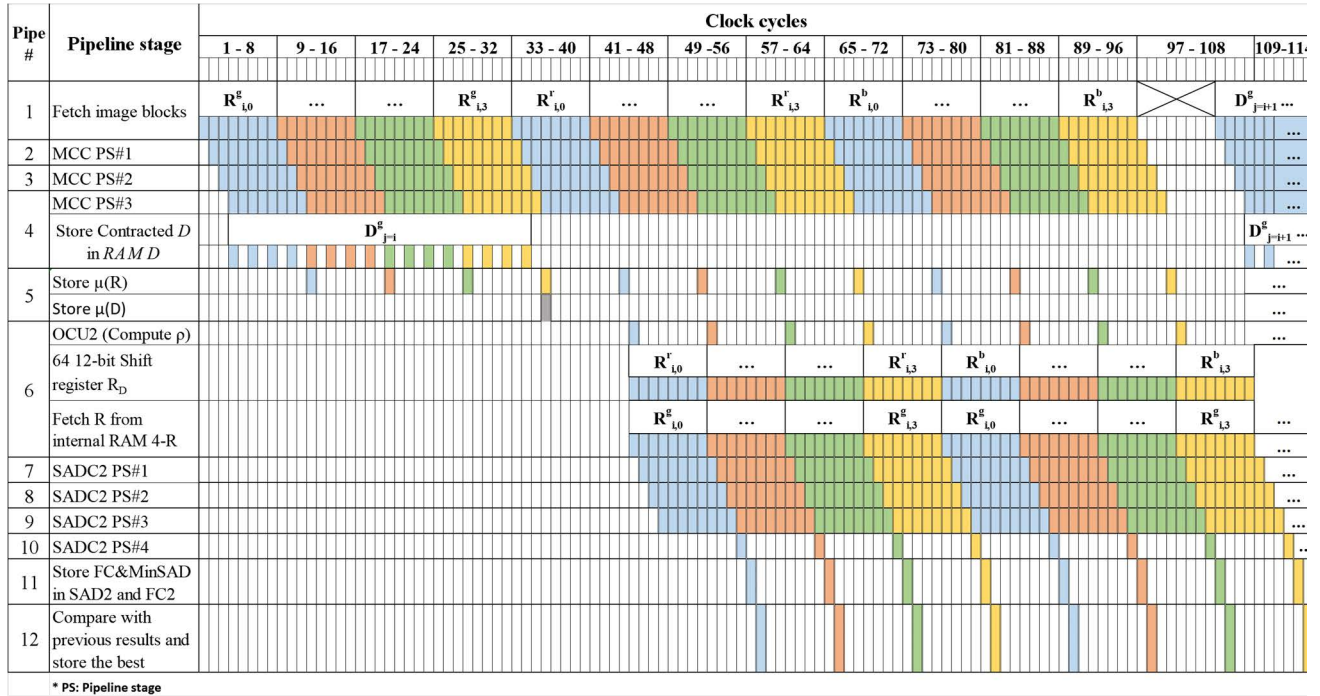

FIGURE 5. MCC unit.

every four-range block $R_{i=j}^g : j = i + 1 \cdots 63$. As mentioned previously, $D_{j=i}^g$ and $R_{i=j}^g$ are constructed, respectively, from the four-range-block $R_{i,p=0,1,2,3}^g$ and the domain block D_j^g, which are matched in *PU1*. $D_{j=i}^g$ and $R_{i=j}^g$ are, respectively, denoted as D_R and R_D in the architecture diagram shown in Fig. 3. Accordingly, the performed matching operations on PU2 do not need any access to the memory and hence the overall memory access is reduced effectively. For each matching, the corresponding processor calculates, for each defined δ value, ρ and SAD. The coefficients corresponding to the lowest SAD value are stored on FC-SAD RAM for further processing.

In order to match $R_{i,p=0,1,2,3}^g$ with $D_{j=i\cdots63}^g$ in *PU1*, the domain blocks $D_{j=i\cdots63}^g$ need to be fetched consecutively from the main memory. However, as the first domain block that needs to be matched (i.e., $D_{j=i}^g$) can be constructed from the range blocks $R_{i,p=0,1,2,3}^g$ which are fetched for encoding R_i^r and R_i^b, therefore there is no need to fetch it again. Thus, MemCtrl fetches only the remaining domain blocks starting from $D_{j=i+1}^g$. In this case, the domain block $D_{j=i}^g$ is constructed by the MCC module at the time of fetching the range blocks $R_{i,p=0,1,2,3}^g$ and the results (d0, d1, d2 and d3 shown in Fig 5 represent four constructed pixels produced every two clock cycles from two consecutive reads, i.e., 2×8 pixel) are stored on the internal RAM D at PU1 processor. Similarly, the mean value of $D_{j=i}^g$ is computed at the same time of computing the mean values of R_i^g and the result is stored in $\mu(D)$ register. Thus, both $R_{i,p=0,1,2,3}^g$ and $D_{j=i}^g$ blocks, and their corresponding mean values are obtained during the encoding process of R_i^r and R_i^b as shown in Fig 6. In this case, the matching process between $R_{i,p=0,1,2,3}^g$ and $D_{j=i}^g$ is

IEEE *Access*

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

Pipe #	Pipeline stage	Clock cycles													
		1 - 8	9 - 16	17 - 24	25 - 32	33 - 40	41 - 48	49 - 56	57 - 64	65 - 72	73 - 80	81 - 88	89 - 96	97 - 108	109-11
1	Fetch image blocks	$R^g_{i,0}$	...	...	$R^g_{i,3}$	$R^r_{i,0}$	...	...	$R^r_{i,3}$	$R^b_{i,0}$	...	...	$R^b_{i,3}$	✕	D^g_{j+1} ...
2	MCC PS#1														
3	MCC PS#2														
	MCC PS#3														
4	Store Contracted D in $RAM\ D$		$D^g_{j=i}$												D^g_{j+1} ...
5	Store μ(R)														
	Store μ(D)														
6	OCU2 (Compute ρ)														
	64 12-bit Shift register R_D							$R^r_{i,0}$	...	...	$R^r_{i,3}$	$R^b_{i,0}$	...	...	$R^b_{i,3}$
	Fetch R from internal RAM 4-R							$R^g_{i,0}$	...	...	$R^g_{i,3}$	$R^g_{i,0}$	...	...	$R^g_{i,3}$
7	SADC2 PS#1														
8	SADC2 PS#2														
9	SADC2 PS#3														
10	SADC2 PS#4														
11	Store FC&MinSAD in SAD2 and FC2														
12	Compare with previous results and store the best														

* PS: Pipeline stage

FIGURE 6. The pipeline processing for encoding the range blocks of the secondary components using the proposed architecture with the searchless scheme.

performed directly after the searchless encoding stage is completed. To be more specific, the matching process starts once RAM 4-R is ready to provide the required range blocks for the PU1 processor. Since RAM 4-R is still in used by PU2 for twelve clock cycles after fetching the last range block from the main memory (see Fig. 6), therefore, this number of clock cycles are required before starting the matching operation (i.e., computing SAD with SADC unit) between $R^g_{i,p=0,1,2,3}$ and $D^g_{j=i}$ in PU1. As ρ is required in the matching operation, therefore, it is computed one clock cycle earlier (clock cycle #108). Thus, for preparing the next domain block D^g_{i+1} for the next matching operation, D^g_{i+1} needs to be fetched four clock cycles earlier (i.e., clock cycle #105). This number of clock cycles is required to compute $\mu\left(D^g_{i+1}\right)$ and g values for the next matching operation and consequently allows to perform the next matching exactly after the previous one is finished. In this case, D^g_{i+1} is fetched eight clock cycles after completely fetching the range block R^b_i and accordingly the search-based encoding stage is considered to start at this time. As a result, the searchless encoding phase needs eight clock cycles along with the clock cycles required for fetching the range blocks R^g_i, R^r_i and R^b_i (i.e., 96 clock cycles), altogether are 104 clock cycles.

At the clock cycle #105, the processor PU1 starts encoding $R^g_{i,p=0,1,2,3}$ by the proposed search scheme as shown in Fig. 7. At this time, the MemCtrl starts fetching the domain blocks $D^g_{j=i+1,...,63}$ for determining the most matched block for each $R^g_{i,p}$. To do so, MCC computes the mean value for each domain block D^g_j and stores the result in μ (D)

register. On the same time, MCC contracts the fetched domain block to the size of the range block and stores the results on RAM D. Since the contracted D^g_i needs to be read four times from RAM D for computing SAD$(R^g_{i,0}, D^g_i)$, SAD$(R^g_{i,1}, D^g_i)$, SAD$(R^g_{i,2}, D^g_i)$, and SAD$(R^g_{i,3}, D^g_i)$ sequentially, while the domain block D^g_{i+1} is being written on it, therefore, RAM D is designed to buffer two consecutive and contracted domain blocks. Thus, the size of RAM D is 16×8-byte. The computing of SAD values starts on the clock cycle #110 once $R^g_{i,0}$ and D^g_i are fetched from the dual-port RAMs 4-R and D, respectively. Once the SAD values for the predefined four scale coefficients δ are computed, SADC then selects the lowest SAD value, *MinSAD*. The *MinSAD* value with the corresponding fractal codes are stored in two registers, i.e., SAD1 and FC1, respectively. The SAD1 value is then compared with the previously stored SAD, i.e., the minimum SAD value obtained from the previous matching operations for the same range block. The result from this comparison is written into FC-SAD RAM of size 256 29-bit by FC-SAD SCtrl unit. The other three range blocks of R^g_i ($R^g_{i,1}$, $R^g_{i,2}$ and $R^g_{i,3}$) are matched consecutively and similarly as $R^g_{i,0}$. The number of cycles per each matching is 8 as shown in Fig. 7. This process is repeated for each domain block $D^g_{j=i+1,...63}$.

While *PU1* is carrying out the assigned matching operations, the matching operations between $R^g_{i=j,p=0,1,2,3} : j = i + 1 \cdots 63$ and $D^g_{j=i}$ are carried out on *PU2*, utilising the same data fetched for *PU1*. In the process, $\mu\left(R^g_{i=j,p}\right)$ and $\mu\left(D^g_{j=i}\right)$ are computed at the same time of computing

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation
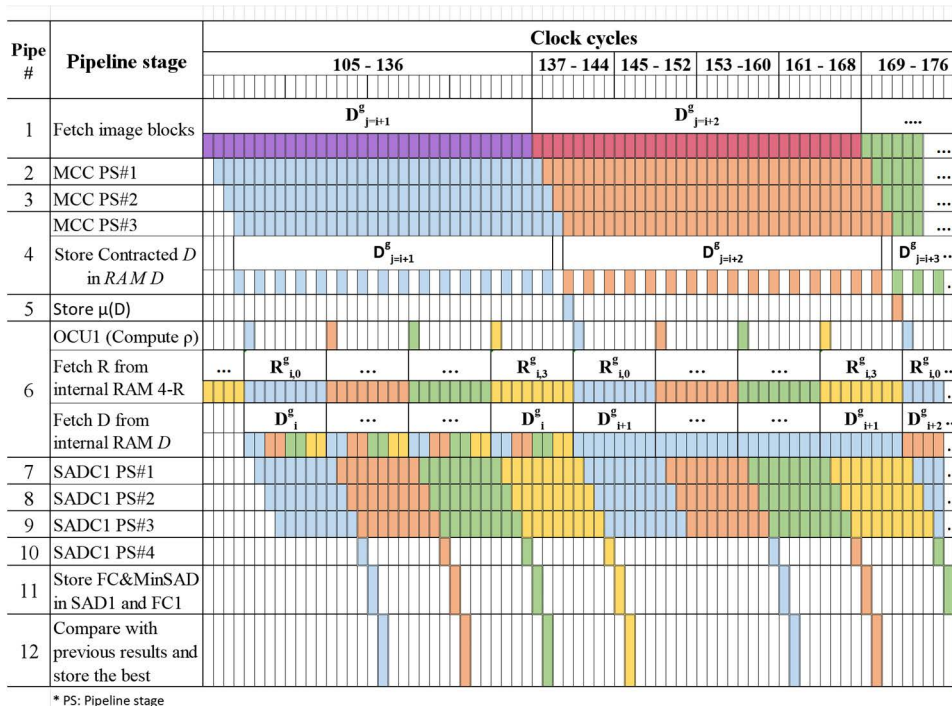
IEEE *Access*

FIGURE 7. Pipelining execution time for encoding the range blocks of the primary component in the processor PU1 using the search scheme.

$\mu\left(D_j^g\right)$ and $\mu\left(R_{i,p}^g\right)$, respectively. The results for $\mu\left(R_{i=j,p}^g\right)$ and $\mu\left(D_{j=i}^g\right)$ are stored in $\mu(R_D)$ and $\mu(D_R)$, respectively. All the mean computations are carried out on the MCC unit. However, unlike the situation in PU1, the mean values of the range blocks $\mu\left(R_{i=j,p}^g\right)$ are calculated after the calculation of the mean value of the domain block $\mu\left(D_{j=i}^g\right)$. Therefore, PU2 starts the matching process once $\mu\left(R_{i=j,p}^g\right)$ is available in $\mu(R_D)$ register. In this case, the first matching of PU2 is performed after fetching the first range block $R_{i=j,0}^g : j = i + 1$ and computing its mean value $\mu\left(R_{i+1,0}^g\right)$. This is corresponding to twelve clock cycles from the beginning of the search encoding stage. Thus, PU2 computes δ_u of $R_{i+1,0}^g$ and $D_{j=i}^g$ using its own OCU unit (i.e., OCU2), and stores the results in their respective register in the following cycle. Following this process, the SADC2 is enabled to calculate the SAD values and needs to be fed with the corresponding range and domain block, i.e., $R_{i+1,0}^g$ and $D_{j=i}^g$. To do so, $R_{i+1,0}^g$ and the contracted $D_{j=i}^g$ are stored in the internal RAMs (i.e., 64 12-bit shift register and dual-port RAM D_R of the size 8×64 bits) to be utilised by SADC2. The outputs from SADC2 are stored on SAD2 and FC2 registers. The SAD2 value is then compared with the minimum SAD value that is already obtained from the previously performed matching operations and stored in FC-SAD RAM. Thus, SAD2 and FC2 values are stored in FC_SAD RAM if and only if the SAD2 value is smaller than the previously stored SAD value. These operations are repeated for each $R_{i=j,p}^g : j = i+1 \cdots 63$.

Fig. 8 demonstrates the pipelining execution time for encoding the primary range blocks in the processor PU2.

In conclusion, the range blocks ($R_{i,p=0,1,2,3}^r$ and $R_{i,p=0,1,2,3}^b$) of the secondary components are encoded first, followed by encoding the range blocks $R_{i,p=0,1,2,3}^g$ of the primary component. At the end of encoding these range blocks with the proposed encoding schemes, the next blocks are fetched and processed in the same manner. This process is repeated until encoding the entire image blocks.

IV. RESULTS AND DISCUSSION
The proposed FCIC architecture, depicted in Fig 3, has been synthesised and implemented on Altera DE4 board. This architecture utilises the inherent correlation between the colour components to compress 1024×1024 RGB image in an efficient short time. The design characteristics are given in Table 1. From this table, it can be seen that the number of logic elements (LEs) utilised for the proposed FCIC design is 4973 LEs, which represents around 2% only of the total available LEs. In respect to the memory requirement, this design needs 13403 bits of memory size. Finally, this design can successfully operate at a maximum frequency of 380 MHz.

Meanwhile, the runtime or the encoding time can be calculated also by computing the time needed to encode each component individually and adding up the results. Since the colour components are processed in two different groups (i.e., primary, and secondary groups), the total encoding time equals the sum of the encoding time for each group. Here, the
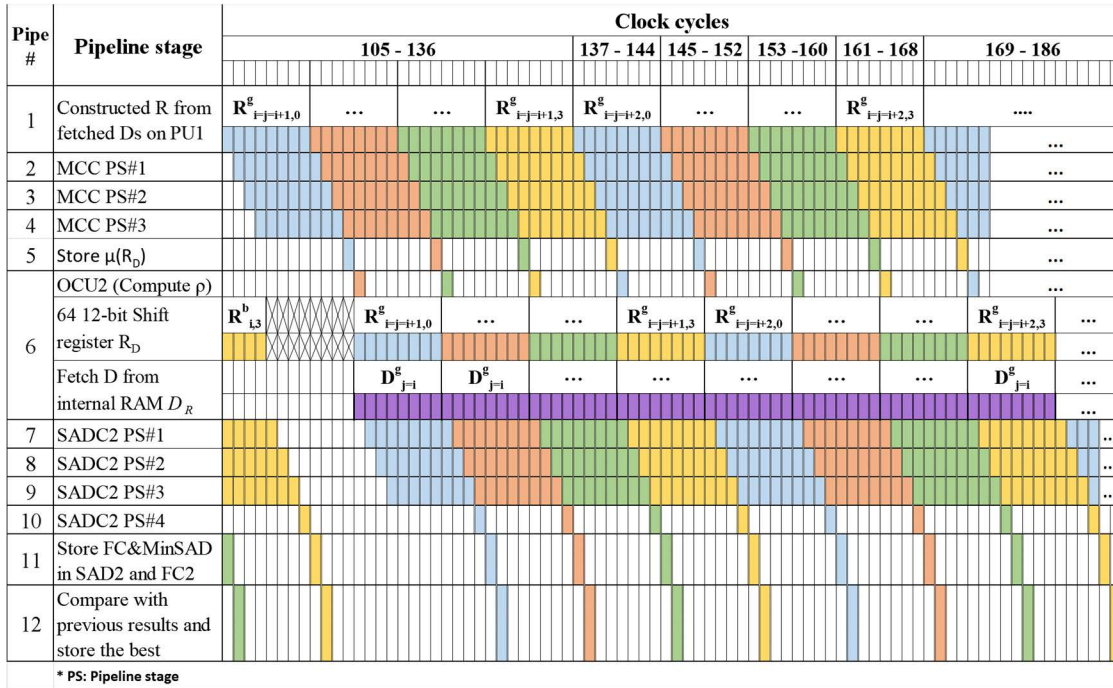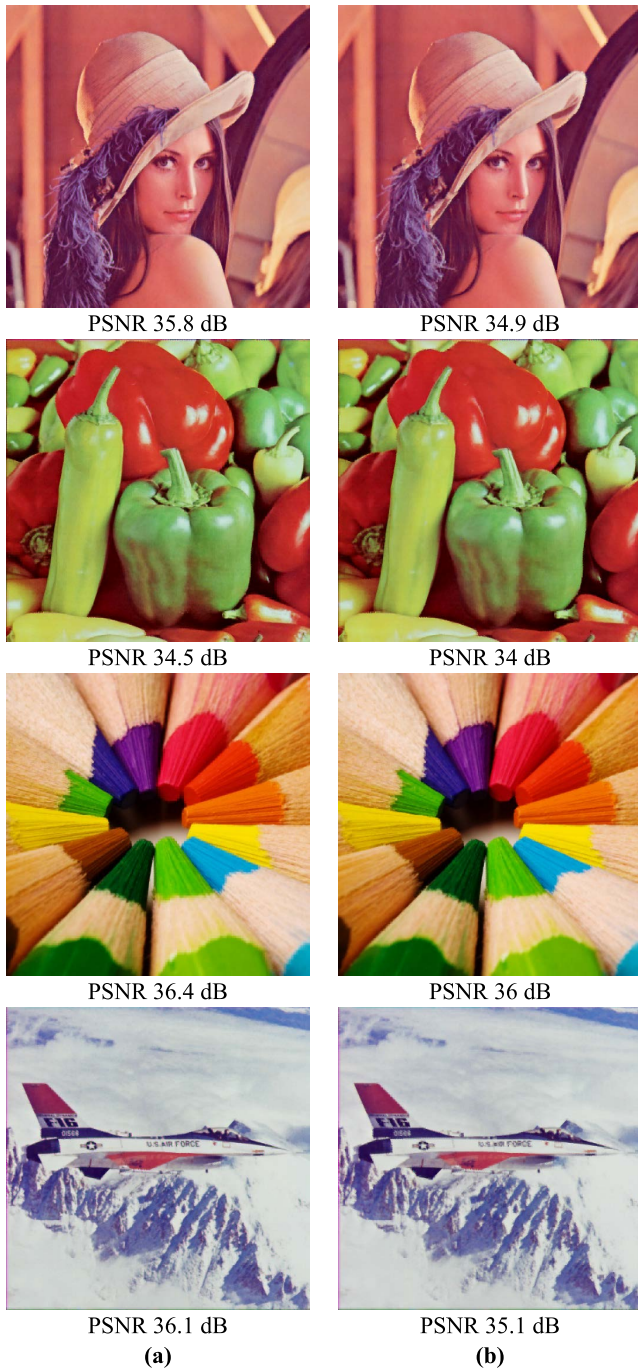
IEEE *Access*

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

Pipe #	Pipeline stage	Clock cycles 105 – 136	137 – 144	145 – 152	153 –160	161 – 168	169 – 186
1	Constructed R from fetched Ds on PU1	$R^g_{i-j=i+1,0}$ ... ... $R^g_{i-j=i+1,3}$	$R^g_{i-j=i+2,0}$	...	...	$R^g_{i-j=i+2,3}$	....
2	MCC PS#1						...
3	MCC PS#2						...
4	MCC PS#3						...
5	Store μ(R_D)						...
6	OCU2 (Compute ρ)						...
	64 12-bit Shift register R_D	$R^b_{i,3}$ $R^g_{i-j=i+1,0}$ ... ... $R^g_{i-j=i+1,3}$	$R^g_{i-j=i+2,0}$	...	...	$R^g_{i-j=i+2,3}$	...
	Fetch D from internal RAM D_R	$D^g_{j=i}$ $D^g_{j=i}$ ...	...	...	...	$D^g_{j=i}$	...
7	SADC2 PS#1						...
8	SADC2 PS#2						...
9	SADC2 PS#3						...
10	SADC2 PS#4						
11	Store FC&MinSAD in SAD2 and FC2						
12	Compare with previous results and store the best						

* PS: Pipeline stage

FIGURE 8. Pipelining execution time for encoding the range blocks of the primary component in the processor PU2 using the search scheme.

TABLE 1. FCIC performance.

Number of LEs	4973 (2% of the total LEs available)
Memory utilization	13403
f_{max}	380 MHz
Encoding time	12.2 ms
Compression ratio	46.5

processing time of each group is obtained by computing the total number of clock cycles that are required to execute the whole process.

Referring to Section 3.2.3, every set of eight 8 × 8 range blocks in the secondary group (i.e., $R^r_{i,p=0,1,2,3}$ and $R^b_{i,p=0,1,2,3}$) needs 104 clock cycles to be processed by the proposed searchless encoding scheme. As the two-colour components of the secondary group have 4096 sets of eight range blocks ($4096 = \frac{2 \times 1024 \times 1024}{8 \times 8 \times 8}$), hence, the total number of clock cycles required for encoding these colour components is $4096 \times 104 = 425984$ cycles. As a result, the encoding time for the secondary group equals $425984 \times \frac{1}{380\ MHz} = 1.121\ ms$. For the other group referred to as the primary component, where the G component is encoded by the introduced search scheme, every four range blocks $R^g_{i,p=0,1,2,3}$ need to be matched with 64 domain blocks. As given also in the same section, the matching operations are carried out on two processors in parallel. Thus, each $R^g_{i,p=0,1,2,3}$ is matched with the domain blocks $D^g_j : j = i, \ldots, 63$ in the first processor PU1
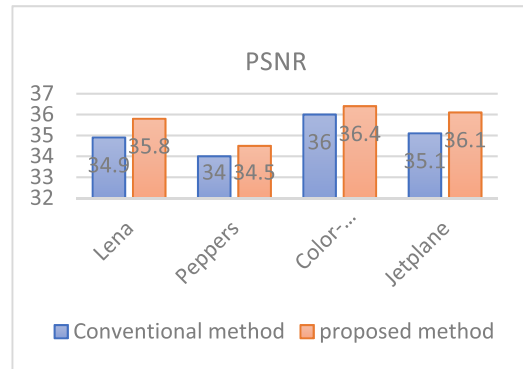
and with $D^g_j : j = 0, \ldots, i - 1$ in the second processor PU2. On general, each processor carries out half of the matching numbers for each range block, i.e., 32 matching operations per each processor. Referring to Figs. 7 and 8, it can be seen that each matching of $R^g_{i,p=0,1,2,3}$ and D^g_j is executed in 32 clock cycles. Therefore, the number of clock cycles for encoding each $R^g_{i,p=0,1,2,3}$ is equal to 32 (the number of matching operations per processor) $\times 32$ (the number of clock cycles per each matching) = 1024 cycles. Two additional clock cycles were required at the end of the matching operations before it can move to the searchless phase. Consequently, each $R^g_{i,p=0,1,2,3}$ requires 1026 clock cycles to be encoded. Thus the primary component requires 4096 (the total number of R^g_i for the entire image) $\times 1026$ (the number of clock cycles for each R^g_i) clock cycles. This corresponds to the encoding time of 11.059 ms for $fmax = 380\ MHz$. Thus, the total time for compressing the entire image is $12.2\ ms = 11.059\ ms$ (the primary component's encoding time) + 1.121 ms (the secondary components' encoding time).

Similarly, the CR can be also calculated based on the sizes of the fractal codes of both the primary and the secondary groups. Since they are encoded with different bit sizes, the CR needs to be computed individually. For the primary group, the fractal code size is 15 bits for each range block (i.e., 2-bit for δ, 7-bit for ρ and 6-bit for the spatial information of the corresponding matched domain block). In this case, the compressed size of the primary group is $\frac{1024 \times 1024}{8 \times 8} \times 15 = 245760$ bits. Unlike the primary group whose range blocks are encoded with three parameters, the secondary

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE *Access*

PSNR 35.8 dB   PSNR 34.9 dB

PSNR 34.5 dB   PSNR 34 dB

PSNR 36.4 dB   PSNR 36 dB

PSNR 36.1 dB   PSNR 35.1 dB

(a)   (b)

FIGURE 9. Decompression results of four well-known RGB images encoded with (a) the proposed approach and (b) the conventional approach.

FIGURE 10. PSNRs of four decoded well-known RGB images encoded by the proposed and the conventional approach.

TABLE 2. Comparison of the performance of the introduced design with the proposed approach vs the conventional approach.

	Introduced design with the proposed approach	Introduced design with the conventional approach
Runtime	12.2 ms (82 fps)	34.2 ms (29 fps)
Average PSNR	35.7	35
CR	46.5	34.1

group's range blocks are encoded with only two parameters (i.e., δ and ρ). In this case, each 8×8 range block in the secondary group is encoded by 9 bits (i.e., 2-bit for δ and 7-bit for ρ). Thus, the compressed size of the secondary group is equal to the total number of range blocks multiplied by 9, which is $\frac{2 \times 1024 \times 1024}{8 \times 8} \times 9 = 294912$ bits. As a result, the total CR of the image is equal to the ratio between the original image size and the total compressed size of both the groups, which is $\frac{3 \times 1024 \times 1024 \times 8}{245760 + 294912} = 46.5$.

In order to obviously see the advantages of utilising the correlation between the colour components when encoding colour images as in the presented design, the result from the introduced design needs to be compared with a similar design but using the conventional approach that encodes each colour component individually. To do so, the presented design has been modified to encode each colour component separately with the introduced search scheme (the searchless encoding scheme is disabled as it cannot be used to encode all the colour components in the image). Thus, all colour components of the RGB image are encoded as similar as the G colour component encoded. By doing so, we can fairly compare the proposed design and see the disadvantages of encoding colour images individually as this is the case for nearly all designs available in the literature. Table 2 shows the comparisons for both approaches. It is clear that the design with the proposed approach is nearly threefold faster than the same design without utilising the correlation among the colour components. With the proposed approach, R and B components are encoded in 0.56 ms each, compared to 11.4 ms when encoded individually by the search scheme. Thus, the proposed approach reduces the compression time to one third of the conventional approach. Furthermore, the compression rate of the proposed approach is also increased by 36 % over the conventional approach as the spatial information for the mapped blocks are not required to be stored when the searchless scheme is used. Although the CR for the proposed approach is increased, the PSNR is also improved by 0.7 dB

TABLE 3. Comparison between the proposed and selected designs.

	Proposed	Jackson et al. [1]	Panigrahy et al. [2]	Haque, et al. [3]	da Rosa Righi, et al. [4]	Ismail, et al. [6]
Implementation	FPGA	APEX20K FPGA	FPGA	GPU	Two dual-core CPUs	GPU
Clock speed (MHz)	380	32.05	75.52	1301	2900	700
Image size	$1024 \times 1024 \times 3$	$512 \times 512 \times 3$	$256 \times 256 \times 3$	$1024 \times 1024 \times 3$	$512 \times 512 \times 3$	$256 \times 256 \times 3$
Search type	Partial Search	Searchless	Full search	Full search	Full search	Partial search
Range block size	Fixed (8×8)	Flexible	Fixed (4x4)	Fixed (4x4)	Fixed (8×8)	Fixed (8×8)
Compression Ratio	46.5	12.8	5.1	4.1	-	-
Encoding time	12 ms	25.2 ms	1.6 s	135 s	25.5 s	1.4 s

on average. The decompressed images for both approaches are shown in Fig. 9 for subjective assessment. For further clarity, PSNR results for the four tested images are shown in Fig. 10. From the illustrated results, it is clear that the introduced design with the proposed approach outperforms the conventional approach significantly.

The achieved performance has also been compared with the performances of the available grayscale-based implementations. The decision to compare with the grayscale-based designs rather than the colour-based designs is due to there is no such appreciated hardware implementation in the literature for encoding colour images, while there are many for encoding grayscale images. One reason for this lack in the colour hardware implementations is that the colour images can be encoded by the one developed for grayscale image too. Thus, several well-known grayscale-based designs and implementations from the literature have been compared and reviewed in Table 3. In this comparison, their reported encoding times for grayscale images have been multiplied by three to obtain the encoding times for RGB images, as each of the three colour components is assumed to be encoded separately as the grayscale image.

From the table, it can be seen that the encoding time of the proposed architecture is the lowest compared to others, which corresponds to $2\times$, $133\times$, $11250\times$, $2125\times$ and $116\times$ less than [1], [2], [3], [4], and [6], respectively. If the image size is taken into account and knowing that the proposed design can encode $512 \times 512 \times 3$ and $256 \times 256 \times 3$ images in 3 ms and 0.75 ms, respectively, the proposed design exhibits $8\times$, $2133\times$, $11250\times$, $8500\times$ and $1866\times$ faster than [1], [2], [3], [4], and [6], respectively. From these figures, it is clear that the proposed architecture is significantly superior to others. Among the targeted platforms, FPGA shows the lowest running times compared to CPU and GPU platforms, even though it runs at a relatively lower clock speed. This is because the hardware solutions are dedicated and optimised for a specific application (fractal-image-compression here) and they process the data in massive processing units in parallel. The high speed in the introduced architecture is attributed to several factors mentioned earlier, including the processing of the image blocks in parallel and in several stages of

pipeline. The later strategy allows to increase the running clock speed to 380MHz, which in turn leads achieving high encoding speed.

In terms of the compression ratio metric, the presented architecture is also superior to others. One of the reasons is the encoding of two-colour components by direct mapping, which results in saving bits allocated for the spatial information. Another reason is the encoding of the image blocks at the size of 8×8 pixels, rather than 4×4 pixels. Although using this block size leads to higher compression rate, it generally causes lower quality in the decompressed images. However, by looking at Fig. 9, it can be seen that the quality of the decompressed images encoded by the proposed approach are clearly good and the PSNRS achieved are above 34 dB for all four tested images.

V. CONCLUSION

As the colour images can be processed by the designs developed for the grayscale images— where each colour component is dealt as a grayscale image and hence is encoded individually— the hardware developers have mainly focused on developing hardware solutions for grayscale images. However, encoding colour images by such a way causes to double the encoding time by n folds, where n is the number of the colour components in the image. To overcome this encoding time problem, it has been developed in this article a high-speed architecture for encoding colour images with fractal technique. The developed architecture utilizes the correlation between the colour components in RGB image in order to encode R and B colour components by direct mapping to G-component (searchless scheme). Thus, only the G component is encoded by the time-consuming search-based scheme. The presented architecture comprises two matching processors. Deep pipeline processing technique with 12 stages is also utilised in developing both processors and results in improving the throughput and the encoding time significantly. The introduced architecture showed the ability to encode 1024×1024 RGB image in 12 ms; where R and B components require only 0.56 ms each compared to 11.06 ms for G component. The proposed design clearly showed the efficiency of utilising inter-colour correlation in developing

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

IEEE *Access*

a specific architecture for encoding colour image by fractal algorithm. As a future work, the inter-frame correlation will be utilized in developing fractal video coding system.

REFERENCES

[1] D. J. Jackson, H. Ren, X. Wu, and K. G. Ricks, "A hardware architecture for real-time image compression using a searchless fractal image coding method," *J. Real-Time Image Process.*, vol. 1, no. 3, pp. 225–237, Mar. 2007.

[2] M. Panigrahy, I. Chakrabarti, and A. S. Dhar, "Low-delay parallel architecture for fractal image compression," *Circuits, Syst., Signal Process.*, vol. 35, no. 3, pp. 897–917, 2015.

[3] M. Haque, A. A. Kaisan, M. R. Saniat, and A. Rahman, "GPU accelerated fractal image compression for medical imaging in parallel computing platform," 2014, *arXiv:1404.0774*.

[4] R. da Rosa Righi, V. F. Rodrigues, C. A. Costa, and R. Q. Gomes, "Exploiting data-parallelism on multicore and SMT systems for implementing the fractal image compressing problem," *Comput. Inf. Sci.*, vol. 10, no. 1, p. 34, 2016.

[5] Y. Fisher, *Fractal Image Compression: Theory and Application*. New York, NY, USA: Springer, 1995.

[6] B. M. Ismail, B. E. Reddy, and T. B. Reddy, "Cuckoo inspired fast search algorithm for fractal image encoding," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 30, no. 4, pp. 462–469, Oct. 2018.

[7] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Trans. Image Process.*, vol. 1, no. 1, pp. 18–30, Jan. 1992.

[8] S. Mozaffari, K. Faez, and M. Ziaratban, "Character representation and recognition using quad tree-based fractal encoding scheme," in *Proc. 8th Int. Conf. Document Anal. Recognit. (ICDAR)*, Aug. 2005, pp. 819–823.

[9] H. Xu, G. Zhai, and X. Yang, "Single image super-resolution with detail enhancement based on local fractal analysis of gradient," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 10, pp. 1740–1754, Oct. 2013.

[10] S. Kiani and M. E. Moghaddam, "A multi-purpose digital image watermarking using fractal block coding," *J. Syst. Softw.*, vol. 84, no. 9, pp. 1550–1562, Sep. 2011.

[11] H.-C. Wu and C.-C. Chang, "Hiding digital watermarks using fractal compression technique," *Fundamenta Informaticae*, vol. 58, pp. 189–202, Jan. 2003.

[12] J. Puate and F. D. Jordan, "Using fractal compression scheme to embed a digital signature into an image," *Proc. SPIE*, vol. 2915, pp. 108–118, Jan. 1997.

[13] B. Wohlberg and G. D. Jager, "A review of the fractal image coding literature," *IEEE Trans. Image Process.*, vol. 8, no. 12, pp. 1716–1729, Dec. 1999.

[14] M. Polvere and M. Nappi, "Speed-up in fractal image coding: Comparison of methods," *IEEE Trans. Image Process.*, vol. 9, no. 6, pp. 1002–1009, Jun. 2000.

[15] C. E. Martin and S. A. Curtis, "Fractal image compression," *J. Funct. Program.*, vol. 23, pp. 629–657, Sep. 2013.

[16] P. Palazzari, M. Coli, and G. Lulli, "Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization," *J. Syst. Archit.*, vol. 45, no. 10, pp. 765–779, Apr. 1999.

[17] S. Liu, W. Bai, N. Zeng, and S. Wang, "A fast fractal based compression for MRI images," *IEEE Access*, vol. 7, pp. 62412–62420, 2019.

[18] M. Ahadullah, S. H. Sapar, N. M. G. Al-Saidi, and M. R. M. Said, "Competitive improvement of the time complexity to encode fractal image: By applying symmetric central pixel of the block," *IEEE Access*, vol. 9, pp. 5028–5045, 2021.

[19] M. Salarian, B. Mohamadinia, and J. Rasekhi, "A modified no search algorithm for fractal image compression," 2015, *arXiv:1501.02894*.

[20] S. Liu, Z. Pan, and X. Cheng, "A novel fast fractal image compression method based on distance clustering in high dimensional sphere surface," *Fractals*, vol. 25, no. 4, Aug. 2017, Art. no. 1740004.

[21] T. Kovács, "A fast classification based method for fractal image encoding," *Image Vis. Comput.*, vol. 26, no. 8, pp. 1129–1136, Aug. 2008.

[22] T. K. Truong, C. M. Kung, J. H. Jeng, and M. L. Hsieh, "Fast fractal image compression using spatial correlation," *Chaos, Solitons Fractals*, vol. 22, no. 5, pp. 1071–1076, Dec. 2004.

[23] C.-C. Wan and C.-H. Hsieh, "An efficient fractal image-coding method using interblock correlation search," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 2, pp. 257–261, Feb. 2001.

[24] K. M. Curtis, G. Neil, and V. Fotopoulos, "A hybrid fractal/DCT image compression method," in *Proc. 14th Int. Conf. Digit. Signal Process. (DSP)*, Jul. 2002, pp. 1337–1340.

[25] D. Saupe, "Accelerating fractal image compression by multi-dimensional nearest neighbor search," in *Proc. DCC Data Compress. Conf.*, Mar. 1995, pp. 222–231.

[26] B. Bani-Eqbal, "Speeding up fractal image compression," in *Proc. IEE Colloq. Fractals Signal Image Process.*, Jan. 1995, p. 2.

[27] D. Saupe and R. Hamzaoui, *Complexity Reduction Methods for Fractal Image Compression*. Munich, Germany: Institut Für Informatik, 1994.

[28] C. S. Tong and M. Pi, "Fast fractal image encoding based on adaptive search," *IEEE Trans. Image Process.*, vol. 10, no. 9, pp. 1269–1277, Sep. 2001.

[29] C. S. Tong and M. Wong, "Adaptive approximate nearest neighbor search for fractal image compression," *IEEE Trans. Image Process.*, vol. 11, no. 6, pp. 605–615, Jun. 2002.

[30] A.-M.-H. Y. Saad and M. Z. Abdullah, "High-speed implementation of fractal image compression in low cost FPGA," *Microprocessors Microsyst.*, vol. 47, pp. 429–440, Nov. 2016.

[31] S. Samavi, M. Habibi, S. Shirani, and N. Rowshanbin, "Real time fractal image coder based on characteristic vector matching," *Image Vis. Comput.*, vol. 28, no. 11, pp. 1557–1568, Nov. 2010.

[32] A.-M.-H. Y. Saad and M. Z. Abdullah, "High-speed fractal image compression featuring deep data pipelining strategy," *IEEE Access*, vol. 6, pp. 71389–71403, 2018.

[33] D. Vidya, R. Parthasarathy, T. C. Bina, and N. G. Swaroopa, "Architecture for fractal image compression," *J. Syst. Archit.*, vol. 46, no. 14, pp. 1275–1291, Dec. 2000.

[34] T. N. Son, T. V. Long, H. M. Thang, and N. T. Dzung, "Efficient implementation of a fractal color image compression on FPGA," in *Proc. Int. Conf. Soft Comput. Pattern Recognit. (SoCPaR)*, Dec. 2013, pp. 184–189.

[35] T. N. Son, O. M. Hung, D. T. Xuan, T. Van Long, N. T. Dzung, and T. M. Hoang, "Implementation of fractal image compression on FPGA," in *Proc. 4th Int. Conf. Commun. Electron. (ICCE)*, Aug. 2012, pp. 339–344.

[36] D. Bereska and H. Palus, "Correlation of colour components of camera output signal and decorrelation methods," *Proc. SPIE*, vol. 4516, pp. 299–306, Aug. 2001.

[37] R. Chaudhari and S. Dhok, "Review of fractal transform based image and video compression," *Int. J. Comput. Appl.*, vol. 57, pp. 23–32, Jan. 2012.

[38] S. S. Selvi and A. Makur, "Variable dimension range and domain block-based fractal image coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 4, pp. 343–347, Apr. 2003.

[39] S. J. Woolley and D. M. Monro, "Optimum parameters for hybrid fractal image coding," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, May 1995, pp. 2571–2574.

[40] A.-M.-H. Y. Saad, M. Z. Abdullah, N. A. M. Alduais, and H. H. Y. Sa'ad, "Impact of spatial dynamic search with matching threshold strategy on fractal image compression algorithm performance: Study," *IEEE Access*, vol. 8, pp. 52687–52699, 2020.

ABDUL-MALIK H. Y. SAAD (Senior Member, IEEE) was born in Jeddah, Saudi Arabia, in 1983. He received the B.Sc. degree (Hons.) in computer engineering from Hodeidah University, Hodeidah, Yemen, in 2006, and the M.Sc. degree in electronic systems design engineering and the Ph.D. degree in digital systems from the Universiti Sains Malaysia, in 2014 and 2018, respectively. His research interests include digital system design, image processing, and AI.

IEEE*Access*

A.-M. H. Y. Saad et al.: Deep Pipeline Architecture for Fast Fractal Color Image Compression Utilizing Inter-Color Correlation

MOHD ZAID ABDULLAH received the B.App.Sc. degree in electronic from the Universiti Sains Malaysia (USM), in 1986, and the M.Sc. and Ph.D. degrees in instrument design and application from the University of Manchester Institute of Science and Technology, U.K., in 1993. He remained with the University of Manchester Institute of Science and Technology, carrying out research in electrical impedance tomography. He joined Hitachi Semiconductor (Malaysia) as a Test Engineer. He is currently a Professor with the School of Electrical and Electronic Engineering, Universiti Sains Malaysia. He has published numerous research papers in international journals and conference proceedings. His research interests include microwave tomography, digital image processing, computer vision, and ultra-wide band sensing. One of his papers was awarded The Senior Moulton medal for the best article published by the Institute of Chemical Engineering, in 2002.

NAYEF ABDULWAHAB MOHAMMED ALDUAIS received the B.Eng. degree in computer engineering from Hodeidah University, Yemen, in 2007, and the master's and Ph.D. degrees in communication and computer engineering from the Faculty of Electrical and Electronic Engineering (FKEE), Universiti Tun Hussein Onn Malaysia (UTHM), Malaysia, in 2015 and 2019, respectively. He is currently a Lecturer and a Researcher in the Internet of Things (IoT) at the Faculty of Computer Science and Information Technology (FSKTM), UTHM, having previously worked as an Assistant Lecturer with the Faculty of Computer Science and Engineering, Hodeidah University, from 2007 to 2013. He has authored numerous papers in journals and conference proceedings. His research interests include WSN and the IoT. He received numerous medals and scientific excellence certificates.

ANTAR SHADDAD HAMED ABDUL-QAWY received the bachelor's degree in computer engineering from Hodeidah University, Yemen, in 2005, the Master of Technology degree in computer science from the University of Hyderabad, India, in 2014, and the Ph.D. degree in electronics and communication engineering (Internet of Things) from Kakatiya University, India, in 2019. He is currently an Assistant Professor of information technology with the Department of Mathematics and Computer Science, and the Dean of the Faculty of Science, SUMAIT University, Zanzibar, Tanzania. He previously worked as an Assistant Lecturer at the Department of Computer Engineering, Faculty of Computer Science and Engineering, Hodeidah University, from 2005 to 2012. His research interests include the Internet of Things, wireless sensor networks, the green IoT, and energy-efficient networks. He is a member of ACM.

ABDULLAH B. NASSER (Member, IEEE) received the B.Sc. degree from Hodeidah University, Yemen, in 2006, the M.Sc. degree from Universiti Sains Malaysia, Malaysia, in 2014, and the Ph.D. degree in computer science (software engineering) from Universiti Malaysia Pahang, in 2018. He is currently a University Lecturer in programming and software engineering with the University of Vaasa, Finland. He is the author of many scientific papers published in renowned journals and conferences. His research interests include optimization algorithms, software testing, and artificial intelligence optimization, including software defect prediction and feature selection.

WAHEED ALI H. M. GHANEM received the B.Sc. degree in computer sciences and engineering from Aden University, Yemen, in 2003, and the M.Sc. degree in computer science and the Ph.D. degree in network and communication protocols from the Universiti Sains Malaysia, in 2013 and 2019, respectively. His research interests include computer and network security, cybersecurity, machine learning, artificial intelligence, swarm intelligence, optimization algorithm, and information technology.

ADNAN HAIDER YUSEF SA'D received the bachelor's degree in computer engineering from Hodeidah University, Yemen, in 2005, the M.Sc. degree in electronic system design engineering from the Universiti Sains Malaysia (USM), Malaysia, in 2015, and the Ph.D. degree in communication from the School of Electrical and Electronic Engineering, USM, in 2019. He is currently an Assistant Professor and the Head of the Cyber Security Department, Al-Razi University, Yemen. His current research interests include communication and embedded systems.

● ● ●