

A Survey of Parallel Algorithms for Fractal Image Compression

Dan Liu

Institute of Nautical Science and Technology, Dalian Maritime University
Dalian, Liaoning 116026, China

Peter K Jimack

School of Computing, University of Leeds
Leeds, LS2 9JT, UK

Abstract: *This paper presents a short survey of the key research work that has been undertaken in the application of parallel algorithms for Fractal image compression. The interest in fractal image compression techniques stems from their ability to achieve high compression ratios whilst maintaining a very high quality in the reconstructed image. The main drawback of this compression method is the very high computational cost that is associated with the encoding phase. Consequently, there has been significant interest in exploiting parallel computing architectures in order to speed up this phase, whilst still maintaining the advantageous features of the approach. This paper presents a brief introduction to fractal image compression, including the iterated function system theory upon which it is based, and then reviews the different techniques that have been, and can be, applied in order to parallelize the compression algorithm.*

1. Introduction

Data compression is a ubiquitous feature of modern society, occurring in many forms and in many settings. The variety of compression techniques is also very diverse, ranging from lossless compression of quantitative data [1, 2] through to lossy compression of diffuse data such as audio, image and movie files [3, 4, 5, 6]. The advantage of lossy compression is that excellent compression ratios may be obtained by ignoring aspects of the data that are unimportant. For example, if the human eye is incapable of distinguishing between two images then it is reasonable to assume that any differences between them are not significant. It follows therefore that, by their nature, lossy compression algorithms tend to be designed with a particular application in mind; image data being a particularly common example.

By far the most popular lossy compression algorithm for photographic types of image is the JPEG algorithm [7]. This is widely used in all areas of image storage and processing, from digital cameras through to photographic archives and libraries. For most purposes the JPEG algorithm is highly satisfactory in that it is fast and efficient, allowing a choice between moderate compression ratio with small loss of quality, through to large compression ratio with a more noticeable loss of quality.

Other image compression techniques are available however [8, 9]. One such approach is based upon fractal image compression (FIC) [9, 22-32]. This technique seeks to exploit affine redundancy that is present in typical images in order to achieve high compression ratios, generally maintaining good image quality with resolution independence. The main drawback of FIC however is that there is a very high computational cost associated with the encoding phase [10]. This is no doubt the main reason that the approach has remained less popular than faster alternatives such as JPEG. Nevertheless, there are a number of reasons why FIC should not be ignored. Firstly, although the encoding is expensive, the decoding is fast and straightforward, and allows smooth images to be recovered at all levels of resolution. Furthermore, the compression ratios achieved can be very high [11] and so if data density or data transmission rates are of more importance than real-time compression then FIC may be advantageous. Finally, as computer hardware continues to increase in speed and decrease in cost, resources that once seemed prohibitive, for encoding large quantities of data, have now become widely accessible.

In the light of the above discussion it seems timely therefore to revisit the FIC approach and consider its application in the context of modern computer architectures. Today's commodity processors are cheap and the combination of fast networking and switching, and portable parallel message passing libraries such as MPI [12], mean that parallel computational environments are easily achievable in terms of both cost and the level of expertise required to maintain and program them. As will be demonstrated below, the fractal image compression algorithm itself is highly amenable to parallel implementation and so in this paper we are motivated to return to the FIC approach with a view to considering the main issues associated

with its application on parallel architectures. This can ensure that the compression phase need not be the bottleneck that it once was, and the positive features of fractal compression will become available without this major drawback.

The purpose of this short paper is therefore to undertake a brief survey of prior research on parallel fractal image compression, and then to draw conclusions on what these techniques can offer in the future. The structure of the rest of the paper consists of an introduction to the iterated function theory that lies at the heart of FIC algorithms, followed by a brief description of the key aspects of FIC methods. This section sets the scene for the main contribution of the paper which is a survey of parallel fractal image compression. The paper then concludes with a discussion of what has been found and how it may be of importance in looking to the future.

2. Iterated Function System Theory and Fractal Image Compression

The basis for Fractal image compression is the construction of an Iterated Function System (IFS) that approximates the original image. An IFS is a union of contractive transformations, each of which maps into itself. Specifically, for a transformation W to be contractive, equation (1) must be satisfied:

$$d(W(P_1), W(P_2)) < d(P_1, P_2). \quad (1)$$

This equation states that the distance $d(P_1, P_2)$ between any two points in a metric space X is reduced by applying the transformation W which maps X into itself. For example, a metric to measure distance when $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are in two dimensional Euclidean space is the standard Euclidean metric, given in (2):

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \quad (2)$$

In practice, for image compression algorithms we will work in a different space, consisting of blocks of pixels, and using an appropriate choice of metric such as that in (5) below.

Provided that a mapping W is contractive, and maps all points of X into X , the Contractive Mapping Fixed Point Theorem holds, proving that there is one and only one attractor for W [13]. As an extension to this, [13], one may show that an IFS also has a unique attractor. Hence by defining an IFS on a space of images, any initial image will converge to one and only one final attractor (which depends only on the IFS that has been defined). With Fractal image compression therefore, the goal is to encode an image as the fixed point of a suitable contractive mapping, which must be constructed.

In order to describe this in a little more detail, consider an arbitrary grayscale image, T , of size $I \times I$ pixels. This image may be partitioned into blocks of pixels in two different ways: the range blocks and the domain blocks. The range blocks, R , are a set of

non-overlapping image blocks of size $k = n \times n$, which are denoted as $\{R_i\}_{i=1}^{N_R}$. The number

of range blocks is $N_R = \frac{I}{n} \times \frac{I}{n}$, and the original image T is the union of $\{R_i\}_{i=1}^{N_R}$:

$$T = \bigcup_{i=1}^{N_R} R_i \quad (3)$$

The domain blocks are also sub-blocks of the original image T , and they must cover the whole image. Unlike the range blocks however, the domain blocks may be overlapping (and usually are). Furthermore, the domain blocks should be larger in size than the range blocks. One way in which the domain blocks may be obtained is by sliding a window of size $l = m \times m$, where $m > n$, throughout the image to construct the domain pool (the set of domain blocks).

To encode a range block R , each of the blocks in the domain pool is scaled to the size of the range block, and is then compared to R with respect to intensity offset and contrast parameters, as well as the eight isometric transformations (the identity, reflections about the mid-horizontal and the mid-vertical axes, reflections about each diagonal, and rotations through 90° , 180° and 270°). The set of contracted domain blocks is denoted as $\{D_i\}_{i=1}^{N_D}$,

where N_D is the number of domain blocks in the domain pool. Each domain block has to be scaled and the eight isometries must then be applied. The resulting pool, C say, of size $8 \times N_D$, is called a codebook pool. It is the domain block which has the closest match with R from the codebook pool which is selected as the best matched block. Details of this match are provided below however, for simplicity, we restrict this introduction to the special case where $N_D = N_R = N$.

The overall form that the contraction takes is

$$T = W(T) = \bigcup_{i=1}^N \omega_i(D_i)$$

where each of the component transformations, ω_i , must be defined. In the case of grayscale images these transformations take the form

$$\omega_i \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \\ o_i \end{pmatrix}$$

where s_i controls the contrast and o_i controls the brightness of the transformation [10].

The specific choice of parameters for the affine transformation ω_i are determined by minimizing the following quantity:

$$R_{MSE} = \|R - s(D + oI)\| \quad (4)$$

Here I denotes the Identity matrix of dimension N , and s and o are the above contrast and offset parameters, respectively, which must be determined in advance (see below) to calculate the distance between D and R . The contrast factor should ensure the contractility of the transformation. The metric $\|\cdot\|$ is the mean square error (MSE) metric. This metric expresses the distance between two images or two range blocks. Assuming two images or image blocks, S and S' , possess \hat{n} pixels with intensities $S_1, S_2, \dots, S_{\hat{n}}$ and $S'_1, S'_2, \dots, S'_{\hat{n}}$, the distance between the blocks can be expressed as (5):

$$\|S, S'\| = \frac{1}{\hat{n}} \sum_{i=1}^{\hat{n}} (S_i - S'_i)^2 \quad (5)$$

The objective when determining s and o is to minimize R_{MSE} . This is true when the partial derivatives of R_{MSE} with respect to s and o are zero. Then, after a little algebra [14, 35], s can be found to be given by (6) and o to be given by (7):

$$s = \frac{N \left(\sum_{i=1}^N D_i R_i \right) - \left(\sum_{i=1}^N D_i \right) \left(\sum_{i=1}^N R_i \right)}{N \left(\sum_{i=1}^N D_i^2 \right) - \left(\sum_{i=1}^N D_i \right)^2} \quad (6)$$

$$o = \frac{\sum_{i=1}^N R_i - s \sum_{i=1}^N D_i}{N} \quad (7)$$

Finally, by substituting (6) and (7) into (4), the distance R_{MSE} can be computed as (8):

$$R_{MSE} = \frac{\sum_{i=1}^N R_i^2 + s \left(s \sum_{i=1}^N D_i^2 - 2 \sum_{i=1}^N D_i R_i + 2o \sum_{i=1}^N D_i \right) + o \left(No - 2 \sum_{i=1}^N R_i \right)}{N} \quad (8)$$

The domain block which results in the smallest value of R_{MSE} is then chosen as the best matched block, and the corresponding parameters for the transformations may be encoded and stored.

Note that it is possible to estimate the computational complexity of the encoding algorithm. Given an $I \times I$ pixel image to encode, where N_D is the number of domain blocks and N_R is the number of range blocks, then the time required to compute the best mapping is given by:

$$\tau_s = 8N_D \cdot N_R \tau_t + 8N_D \cdot N_R \tau_d + N_D \cdot (8N_R - 1) \tau_c. \quad (9)$$

Here τ_t is the time to perform a transformation, τ_d is the time to find the distance between two blocks, and τ_c is the time to compare two distances. Since N_D and N_R are both of the order of I^2 , the overall computing time is at least $O(I^4)$.

It is important to notice that each mapping exploits self-similarities that are present in most images at different scales. Thus, images with random content are not likely to be compressed very well as only few similarities of different size are likely to exist. Of course most images are not random and so the FIC approach tends to be much more effective. Not surprisingly there exists a significant amount of work that has been undertaken to develop variants on the basic algorithm above in order to improve performance [10, 11, 15, 16]. Generally, the attempts to speed up the fractal encoding consist of modifying the following aspects [11]: the composition of the domain pool, the type of search used in block matching, or the representation/quantization of the transform parameters.

One of the most important techniques that is applied in conjunction with the basic algorithm described above, is the use of the quad-tree data structure [10]. The basic idea is that we begin with quite a small range pool and a small domain pool and seek to find a satisfactory match to each range block from the set of domain blocks. If the mean square error is less than a desired tolerance then the match is deemed to be satisfactory but if it is not then the range block is split into four new range blocks, based upon its four quadrants. This process is repeated recursively until a sufficient quality is obtained. The main advantage of the approach is that it minimizes the number of range blocks required by having them as large as possible within any given region of the image. As we will see below however, when a parallel implementation of the algorithm is required, it adds considerable complexity to the load balancing across the processors.

Other research in Fractal image compression concentrates on investigating different transformations and improving search algorithms for matching transformations, each with the aim of decreasing the compression time [17, 18]. An additional technique that many fractal compression algorithms employ to decrease their execution time is to use a classification scheme, typically based upon simple statistics of the blocks. For example, a sub-image may be classified according to its average grey-value and the variance of its four quadrants. Firstly, the sub-image is divided into its four quadrants which are numbered. The pixel values in the

quadrant i of block r are $r_1^i, r_2^i, \dots, r_n^i$ for $i = 1, 2, 3, 4$. Secondly, the sum of the grey values

in a quadrant $A_i = \sum_{j=1}^n r_j^i$ is computed for each quadrant. Finally, a variance

$V_i = \sum_{j=1}^n (r_j^i)^2 - A_i^2$ is computed for each quadrant. Members of the domain pool are then

classified by their average and variance values and only a subset of these classes are searched for a match with each block in the range pool. The aim is of course to improve the speed without significantly affecting the quality of the compressed image. Apart from these attempts focusing on the coding speed, some hybrid coders, such as wavelet-based and DCT-based fractal encoders, have been developed [19, 20]. Various FIC side applications are now also explored in other fields such as image database indexing [21] and even face recognition [17].

3. Parallel Fractal Image Compression

A large number of different strategies have been considered for parallelization of the encoding stage of the fractal image compression algorithms outlined in the preceding section. The interest in using parallel computer architecture stems from the very high computational cost associated with fractal-based compression strategies. As has been explained above, for an $I \times I$ pixel image, the cost of compression is $O(I^4)$ and so the goal of a parallel implementation should be speed up the compression substantially. The decompression algorithm is considerably faster to execute and so parallelization of this is of less importance: although this has received a small amount of attention in the literature.

There are a number of papers that have attempted categorize and summarize the available approaches to parallel implementation of fractal compression. Perhaps the most noteworthy of these are [45, 49, 53], which each provide a description of a variety of parallelization techniques which are grouped together in different classes by these authors. In this review, our aim is not only to update these previous summaries, but also to present the available choices in the context of current computer architectures, which have developed significantly since the work of [49] for example. In addition, although heavily influenced by [45, 49, 53] our categorization is slightly different to any one of these. Finally, we will provide some thoughts on possible alternative parallel variants that are likely to be suited to modern parallel computing environments.

3.1 Classification by granularity

A very simple view of parallel algorithms typically classes them as being either “fine grained” or “coarse grained”. Interestingly however, for parallel fractal image compression, a range of algorithms have been developed over time that have far more than just two different

levels of granularity. At one extreme [52] proposes using n^2 processors for an $n \times n$ pixel image, with each processor working with a single pixel of the image. This is extremely fine-grained parallelism, where expressions such as the sums appearing in equations (6) or (7) are computed in parallel. At the other end of scale, the image can be partitioned amongst the available processors for each processor to then simultaneously apply the sequential fractal compression algorithm on its own sub-image without any communication [45]. This embarrassingly parallel algorithm is extremely coarse grained but suffers from the drawback that the quality of the compressed image will be inferior to that obtained using the sequential algorithm. This is because each processor is only working with a subset of the complete domain pool and so will not generally get as good a match with its range blocks as will be obtained using the sequential algorithm. For the remainder of this review we will only consider parallel algorithms which are able to reproduce the compressed images that are obtained using accepted sequential algorithms.

Intermediate levels of granularity may be found in the work of other authors. For example, [35, 45, 48] present descriptions of some relatively fine-grained algorithms. These involve individual processing units being responsible for calculations on small sets of pixels. For example “pixel processors” store 4×4 pixel maps in [48]. Coarser grained algorithms tend to partition the range pool amongst the processors and then require each processor to find a suitable match to its own subset of the range pool from all of the available domains, e.g. [36, 39, 41, 53]. In some cases the entire domain pool is directly available to each processor and in others it must be partitioned and communicated: this issue is discussed in more detail in section 3.2 below. A different slightly less coarse grained algorithm is presented in [44]. However, there it is assumed that the number of available processors is equal to the number of range blocks and the domain blocks are passed through the system in a pipeline, or systolic architecture.

3.2 Classification by load-balancing algorithm

In this section of the paper we describe the available parallel algorithms in terms of the techniques that they use to ensure that the parallel load is equally distributed across the available processing elements. These are described in terms of three general classes of algorithm however, as we will explain, these classes are not disjoint. We begin with a discussion of pipeline algorithms then discuss static load balancing based upon a partition of the range pool, and finish with a discussion of dynamically partitioning the range pool.

The systolic architecture approach of [44] has already been introduced above. In this approach each range block is compared with a different domain block at any given step. Once the comparison step is completed the domain blocks are shifted to the next processor in the pipeline, for another comparison step. When all domain blocks have passed through the pipeline (which is actually a ring, so as to save any start-up overhead) the comparison step is complete for all range blocks. A similar technique is also adopted in [43]. Another pipeline approach is described in [48], which focuses on the design of a parallel image processing

architecture. In this case the pipeline is used to enhance the performance of the arithmetic in calculating the terms in the expressions (6) and (7) using fine-grained parallelism.

Perhaps the simplest possible load-balancing approach that can be used is to equally partition the range pool across the available processors when using a coarse grained parallel algorithm, this is described in [41, 45] for example. If the time required to find the matching element of the domain pool is the same for each element of the range pool then each processor will automatically have an equal amount of computational work in the crucial matching stage. This is the case when the range and domain pools are fixed and we always seek the best possible match from the domain pool to each element of the range pool. In practice however, the most efficient fractal compression algorithms do not work like this since they either involve use of an adaptive quad-tree (for when no suitable matching domain block is found) or a matching tolerance (such that the search through the domain blocks stops once a suitably close match has been found), or both. This means that the time required to match a domain block to a given range block is not known in advance, and so static partitioning is likely to be unreliable.

As a consequence of this, the majority of coarse-grained parallel algorithms that have been developed tend to use some form of dynamic load-balancing, e.g. [38, 39, 49, 53]. Typically this approach involves a master process and a large number of slave processes. For example, in [39] a master process divides the image equally amongst the slaves. Each slave is therefore responsible for its own subset of the overall domain pool. The master then transmits range blocks to any idle slaves and waits for them to return the best match from their subset of the domain pool. Depending upon whether this is a satisfactory match or not this range block may or may not be sent to another processor. Once all of the range blocks have been processed the master can then move onto the next level of the quad-tree if this is necessary. The master process ensures that all of the slaves are kept busy until the task is completed. This is typical of the master-slave approach to dynamic load balancing. In more general terms, the master keeps a queue of computational tasks and a queue of idle slave processes. Whenever the process queue is not empty the master seeks to give the next job on the task queue to the next process on the idle slave queue.

Interestingly, some dynamic load balancing algorithms have successfully combined the master-slave paradigm with the use of pipelines. For example, in [43] the domain blocks are initially distributed amongst the slaves. Each slave is then assigned a range block by the master. Once the quality of the match between the range block and the subset of the domain blocks has been found on each slave one of two things happens. If the match on a slave is below the desired threshold, or if the range block has already been considered by all of the other slaves, the range block is returned to the master along with a note of the best matching domain block. Otherwise the slave passes the range block on to the next slave in the pipeline of slaves. Before the next matching step begins the master sends a new range block from its task queue to any slave processors who did not receive a range block from their predecessor in the pipeline.

3.3 Classification by data partition

This appears to be the most widely used mechanism in the literature for classifying different coarse grained parallel algorithms, as in [35, 38, 42, 45, 47, 49, 50, 53] for example. Essentially, there are two main classes that may be considered: either each processor has sufficient memory to store a copy of the entire image, and therefore can access the entire domain pool without communication, or it does not. The former case either occurs when a shared memory parallel architecture is used, e.g. [47], or when a distributed memory architecture has a significant amount of memory available on each processor. In practice, in recent years memory technology has scaled sufficiently well to ensure that today's distributed memory architectures will always have sufficient memory on each processor to store even a very high resolution image. This has not always been the case however, and so numerous publications in this field have focused upon algorithms for which the domain pool is partitioned across some or all of the processing elements. Examples include [38, 39, and 43]. These are not discussed in any detail here however since we believe them to be of limited value when sufficient memory is available on each processor. This view is consistent with that expressed in [45], where the replication of the entire image on each processor is found to give the best scalability.

The approach of allowing each processor to have access to the entire domain pool is also used in [49], for example, where it is again found to be beneficial. In fact, [49] describes a number of algorithms that use this approach successfully. Parallelism is achieved by partitioning the range pool across the processing elements (either statically or dynamically, as described above) and so only a very limited amount of inter-processor communication is required.

3.4 Modifications to the standard compression algorithms

So far we have only discussed the parallel implementation of "standard" fractal compression algorithms. Typically these require each range block to be compared against the image of all domain blocks in the domain pool. If a satisfactory match is not found an adaptive quad-tree data structure will also be used. There are however a number of techniques that have been developed in an attempt to improve the speed of the sequential fractal compression algorithms without significantly reducing the quality of the compressed image. These complexity reduction schemes do have an effect on the quality of the compressed image however they are designed with the aim of minimizing this effect. A number of such techniques are described in [11]. Current interest in this topic appears to be as active as ever, e.g. [54, 55, 56].

A number of authors have attempted to combine such complexity reduction schemes with their parallel algorithms, as discussed in [50] for example. Typically these schemes are based upon the use of classification techniques such as that outlined in section 2, where the elements of the range and domain pools are assigned to different classes and range elements of a given class are only compared against the images of domain blocks from the same class. For example, in [37], this block classification approach is used to create 72 different classes based

upon the mean and variance values of the grey values in each block. Many other discrete feature methods exist however and can be incorporated within parallel algorithms, [50].

3.5 Related parallel algorithms

Although this short survey has focused primarily on parallel algorithms for fractal image compression, it should be noted that many of the techniques discussed extend directly to the compression of video. This is well demonstrated in [51] for example, where a pool of three-dimensional range blocks (referred to as “range cubes”) is created by considering sequences of video images together. For example, a sequence of 16 images of 512×512 pixels may be broken down into 8192 range cubes of size $8 \times 8 \times 8$ pixels. The parallel algorithms then partitions these range cubes across the available processors.

Although less important than parallel image and video compression, it is also relevant to note at this point that a number of parallel algorithms have been proposed for fractal image decoding too. Examples include [33, 34, 46] although these are beyond the scope of this particular paper. Clearly the benefits from parallel fractal image decoding are not likely to be so great as with the coding algorithm, since the decoding algorithm is so much faster in the first place: this is one of the main attractions of fractal image compression after all.

4. Discussion

In this short communication we have attempted to review a number of the key ideas and algorithms that have been developed for parallel fractal image compression over the past decade or more. The motivation behind the use of parallel computer architectures in this context comes from the extremely high computational cost of the standard sequential encoding algorithms. A number of available techniques have been discussed from the viewpoints of granularity, load balancing, data partitioning and complexity reduction. In discussing these issues it becomes apparent that a significant amount of care and thought has gone into developing and implementing algorithms suitable for a variety of hardware platforms. It is worth observing however that many of the properties of today’s parallel hardware are not consistent with assumptions made in some of the past work. In particular, it is inconceivable today that a parallel processor would not have sufficient of its own primary memory to store its own copy of the entire uncompressed image (or even a sequence of uncompressed images in the case of video compression).

For this reason, any future focus on improving parallel performance must surely assume that each process has access to the entire domain pool, as discussed in [45, 49] for example. An important issue that will arise in undertaking this work is that of load balancing. As discussed in the previous section, dynamic load balancing is likely to yield the best performances and so it may be that there is benefit to be gained from focusing on this issue more deeply. Typically, for the fractal image compression algorithms considered here, dynamic load balancing has been achieved through the master-slave paradigm. Whilst this is certainly a reliable and straightforward approach it does suffer from the overhead of requiring

the master process to execute in addition to the slaves: which are doing all of the “useful” work. There may therefore be some benefits in considering alternative dynamic load balancing strategies, such as those based upon asynchronous diffusion [57, 58] for example. A possible approach could be to allow each processor to have a copy of the entire image and then allocate itself (based upon its process number for example) an equal share of the range pool. As each processor works through its range blocks, successfully matching them with images of domain blocks, it communicates its remaining number of range blocks to process to its neighboring processors asynchronously. If the remaining load on any processors gets out of balance with its neighbors then the ownership of some of the remaining range blocks can be passed between processors to balance the remaining work dynamically. [59] provides a review of parallel dynamic load balancing algorithms, including this diffusion approach.

It is clear that fractal image compression is unlikely to be as widely used as JPEG is for the general representation of photographic images. However it can play an important role for specific problems where compression ratio is more important than real-time compression, or where it is desirable to reproduce an image of realistic quality regardless of the resolution to which it is viewed. In these cases parallel FIC algorithms have a valuable role to play and efficient, portable implementations should be possible on modern computer architectures.

REFERENCES

- [1] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the Institute of Radio Engineers*, 40 (9): 1098-1101, 1952.
- [2] J. Ziv, A. Lempel, "Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, 23 (3): 337-343, 1977.
- [3] Video codec for audio visual services of Px64 Kbits/s. CCITT Recommendation H. 261, Aug. 1990.
- [4] Digital Compression and Coding of Continuous-Tone Still Images Part 1, Requirements and Guidelines. ISO/IEC JTC1 Committee Draft 10918-1.
- [5] Digital Compression and Coding of Continuous-Tone Still Images Part 2, Compliance Testing. ISO/IEC JTC1 Committee Draft 10918-2.
- [6] M. L. Liou, "Overview of the Px64 Kbits/s Video Coding Standard", *Communications of the ACM*, Vol.34, No.4, 1991.
- [7] Gregory K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of the ACM*, Vol.34, No.4, 1991.
- [8] .S. G. Mallat, "Multifrequency Channel Decomposition of Images and Wavelet Models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.37, No.12, 1989.
- [9] A. E. Jacquin, "Fractal Image Coding: A Review", *Proceedings of the IEEE*, Vol.81, No.10, 1993.
- [10] Y. Fisher, "Fractal Image Compression: Theory and Application". New York: Springer-Verlag New York, Inc., 1995.
- [11] B. Wohlberg and G. d. Jager, "A Review of the Fractal Image Coding Literature", *IEEE Transaction on Image Processing*, vol. Vol. 8, 1999.
- [12] J. DONGARRA, D. WALKER, E. LUSK, et al, "Special Issue - MPI - A Message-Passing

- Interface Standard", *International Journal of Supercomputer Applications and High Performance Computing*, 8 (3-4): 165-&, 1994.
- [13] M. F. Barnsley, "Fractal Everywhere", *Academic Press Inc.*, 2nd Edition, 1993.
- [14] Zhao Erjun, Liu Dan, "Fractal Image Compression Methods: A Review", *The Third International Conference on Information, Technology and Applications*, ICITA2005(Sydney), pp.756-759, 2005.
- [15] R. Distasi, M. Nappi, and D. Riccio," A Range/Domain Approximation Error-Based Approach for Fractal Image Compression", *IEEE Trans. Image Process.*, vol. 15, No.1, pp. 89-97, 2006.
- [16] Y. Iano, d. S. Silvestre, and A. L. M. Cruz," A Fast and efficient Hybrid Fractal-Wavelet Image Coder", *IEEE Trans. Image Process.*, vol. 15, No.1, pp. 98-105, 2006.
- [17] H. E. Komleh, V. Chandran, and S. Sridharan," Face Recognition Using Fractal", *Proc. Int. Conf. Image Processing*, Vol. pp. 58-61, 2001.
- [18] J. M. Beaumont," Image Data Compression Using Fractal Techniques", *BT Techn. Journal*, vol. 9, pp. 93-109, 1991.
- [19] G. Melnikov and A. K. Katsaggelos," A Jointly Optimal Fractal/DCT Compression Scheme", *IEEE Trans. on Multimedia*, vol. 4, No.4, pp. 413-422, 2002.
- [20] G. M. Davis," A Wavelet-Based Analysis of Fractal Image Compression", *IEEE Transaction on Image Processing*, USA, Vol. No. 2, pp. 1998.
- [21] R. Distasi, M. Nappi, and M. Tucci," FIRT: Fractal Indexing with Robust Extensions for Image Databases", *IEEE Trans. Image Process.*, vol. 12, No.3, pp. 373-384, 2003.
- [22] H. Hartenstein, M. Ruhl, and D. Saupe," Region-based fractal image compression", *Image Processing, IEEE Transactions on*, vol. 9, pp. 1171-1184, 2000.
- [23] K. Belloulata and J. Konrad," Fractal image compression with region-based functionality", *Image Processing, IEEE Transactions on*, vol. 11, pp. 351-362, 2002.
- [24] B. Wohlberg and G. d. Jager," A Review of the Fractal Image Coding Literature", *IEEE Transaction on Image Processing*, Vol. No. 12, pp. 1999.
- [25] M. Polvere and M. Nappi," Speed-up in Fractal Image Coding - Comparison of Methods", *IEEE Trans. Image Process.*, Vol. 9, pp. 1002-1009, 2000.
- [26] C. S. Tong and M. Pi," Fast Fractal Image Encoding Based on Adaptive Search", *IEEE Trans. Image Process.*, Vol. 10, pp. 1269-1277, 2001.
- [27] J. Li, G. Chen, and Z. Chi," A Fuzzy Image Metric with Application to Fractal Coding", *IEEE Trans. Image Process.*, Vol. 11, pp. 636-643, 2002.
- [28] C. S. Tong and M. Wong," Adaptive Approximate Nearest Neighbor Search for Fractal Image Compression", *IEEE Trans. Image Process.*, Vol. 11, pp. 605-615, 2002.
- [29] R. Rinaldo and G. Calvagno," Image Coding by Block Prediction of Multiresolution Subimages", *IEEE Trans. Image Process.*, Vol. 4, pp. 909-920, 1995.
- [30] D. J. Duh, J. H. Jeng, and S. Y. Chen," DCT-based Simple Classification Scheme for Fractal Image Compression", *Image and Vision Computing*, Vol. 23, pp. 1115-1121, 2005.
- [31] Y. Iano, D. S. Silvestre, and A. L. M. Cruz," A Fast and Efficient Hybrid Fractal-Wavelet Image Coder", *IEEE Trans. Image Process.*, Vol. 15, pp. 98-105, 2006.
- [32] I. Her," Geometric Transformations on the hexagonal Grid", *IEEE Transaction on Image Processing*, vol. 4, 1995.
- [33] H.T. Chang, and C.J. Kuo, "Intrinsic parallel random iteration algorithm for fractal image

- decoding based on iterated function system code", *Optical Engineering* 44(3), 2005.
- [34] L. Dedera, and J. Chmurny, "A parallel approach to image decoding in the Fractal image block coding scheme".
- [35] U. Erra, "Toward real time fractal image compression using graphics hardware", *Advances in Visual Computing, proceedings lecture notes in computer science 3804:723-728*, 2005.
- [36] J. Hammerle, and A. Uhl. "Approaching Real-time Processing for Fractal Compression".
- [37] J. Hammerle, and A. Uhl. "Classification based speed-up methods for fractal image compression on multicomputers". *Parallel Computation Lecture Notes in Computer Science 1557:276-285*. 1999.
- [38] C. Hufnagl, and A. Uhl. "Algorithms for Fractal Image Compression on Massively Parallel SIMD Arrays". *Real-Time Imaging* 6. 267-281. 2000.
- [39] D. J. Jackson, and T. Blom. "A parallel fractal image compression algorithm for hypercube multiprocessors". *27th Southern Symposium on System Theory*. 274-278. 1995.
- [40] D. J. Jackson, and C. W. Humphres. "A simple yet effective load balancing extension to the PVM software system". *Parallel Computing* 22. 1647-1660. 1997.
- [41] D. J. Jackson, and G. S. Tinney. "Performance analysis of distributed implementation of a fractal image compression algorithm". *Concurrency-Practice and Experience* 8 (5): 357-386. 1996.
- [42] J. Hammerle, and A. Uhl. "Improving the Efficiency of Parallel Fractal Compression using Localized Domain-pools".
- [43] D. J. Jackson, and W. Mahmoud. "Parallel Pipelined Fractal Image Compression using Quadtree Recomposition". *The Computer Journal*. Vol. 39 (1): 1-13. 1996.
- [44] S. Lee, S. Omachi, and H. Aso. "A parallel architecture for quadtree-based fractal image coding". *Proceedings of 2000 International Conference on Parallel Processing*. 15-22.2000.
- [45] P. Palazzari, M. Coli, and G.Lulli. "Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization". *Journal of Systems Architecture* 45. 765-779.1999.
- [46] S. C. Pei, C. C. Tseng, and C. Y. Lin. "A parallel decoding algorithm for IFS codes without transient behavior". *IEEE Transactions on Image Processing*. Vol. 5 (3). 411-415. 1996.
- [47] A. Pommer. "Fractal Video compression on shared memory systems". *Parallel Computation Lecture Notes in Computer Science 1557: 317-326*. 1999.
- [48] K.P.Acken, H.N.Kim, M.J.Irwin and R.M.Owens,"An architecture design for parallel fractal compression", *Application Specific Sysytems, Architectures and Processors*, 1996.
- [49] A. Uhl, and J. Hammerle. "Fractal Image Compression on MIMD architectures I: Basic Algorithms". *The First International Conference on Visual Information Systems*. 1996.
- [50] J. Hammerle, and A. Uhl. "Fractal Image Compression on MIMD architectures II: Classification Based Speed-up Methods". *The Fourth International Conference of the ACPC*. 1999.
- [51] M. Q. Wang, Z. H. Huang, C. H. Lai. "Matching search in fractal video compression and its parallel implementation in distributed computing environments" *Applied Mathematical Modelling* 30 (2006) 677-687. 2006.
- [52] X. Min, T. Hanson, and A. Merigot. "A massively parallel implementation of fractal image compression". *IEEE International Conference on Image Processing*. 1994.
- [53] Z. P. Bodo. "Maximal processor utilization in parallel quadtree-based fractal image

- compression on MIMD Architectures”. *Informatica*. Volume XLIX (2). 2004.
- [54] X. W. Wu, D. J. Jackson, and H. C. Chen. “A fast fractal image encoding method based on intelligent search of standard deviation”. *Computers and Electrical Engineering* 31: 402-421. 2005.
- [55] C. G. Zhou, K. Meng, and Z. L. Qiu. “A fast fractal image compression algorithm based on average-variance function”. *IEICE TRANS. INF. & SYST.* Vol. E89-D (3). 2006.
- [56] R. Distasi, M. Nappi, and D. Riccio. “A range/domain approximation error-based approach for fractal image compression”. *IEEE Transactions on Image Processing*. Vol. 15 (1). 2006.
- [57] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors", *J. of Parallel and Distributed Computing*, 7, 279-301, 1989.
- [58] G. Horton, "A Multi-Level Diffusion Method for Dynamic Load Balancing", *Parallel Computing*, 19, 209-218, 1993.
- [59] N. Touheed, P. Selwood, P. K. Jimack, & M. A. Berzins, "Comparison of Some Dynamic Load-Balancing Algorithms for a Parallel Adaptive Flow Solver", *Parallel Computing*, vol.26, pp.1535--1554, 2000.