Doctor of Engineering Thesis

Parallel Processing Architecture for Fractal Image Compression

Department of Electrical and Communication Engineering
Graduate School of Engineering
Tohoku University, JAPAN

Shinhaeng Lee
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>8</td>
</tr>
<tr>
<td>Acronyms</td>
<td>10</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>1.1 Background and Motivations</td>
<td>11</td>
</tr>
<tr>
<td>1.2 Purpose of This Thesis</td>
<td>15</td>
</tr>
<tr>
<td>1.3 Organization of This Thesis</td>
<td>18</td>
</tr>
<tr>
<td>2 Fractal Image Compression</td>
<td>20</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>20</td>
</tr>
<tr>
<td>2.2 Theoretical Foundations</td>
<td>22</td>
</tr>
<tr>
<td>2.3 Fractal Image Coding Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>2.4 Fractal Image Decoding Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>3 Alternating Binary-tree Partitioning (ABP) for Fractal Image</td>
<td>35</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>35</td>
</tr>
<tr>
<td>3.2 Conventional Partitioning Schemes of FIC</td>
<td>36</td>
</tr>
</tbody>
</table>
CONTENTS

3.3 Binary-tree Partitioning ........................................... 38
3.4 Alternating Binary-tree Partitioning (ABP) ...................... 40
3.5 Experimental Results ............................................. 42
3.6 Summary ................................................................... 45

4 An Architecture for Fixed-Size Partitioning Fractal Image Com-
pression ........................................................................ 48
4.1 Introduction ............................................................. 48
4.2 The Proposed FPFIC architecture ................................... 49
   4.2.1 The Extraction of the Domain Pool: Stage I ........... 51
   4.2.2 Determining the Best Domain Block: Stage II ....... 53
   4.2.3 The Fast Comparison Module (FCM) ..................... 54
   4.2.4 Validity of Fast Comparison Operation ................. 62
4.3 For the P-pixels-interval Domain Pool ............................ 68
4.4 Experimental Results ............................................... 69
4.5 Summary ................................................................... 72

5 An Architecture for Quadtree-based Partitioning Fractal Image
Compression ..................................................................... 75
5.1 Introduction ............................................................. 75
5.2 The Proposed QPFIC architecture ................................. 76
   5.2.1 Generation of $\mathcal{D}^{d_{max}}$ and the MAD calculation for $\mathcal{D}^{d_{max}}$
       (First Part of Phase 1) ............................................. 77
   5.2.2 The MAD calculation for $\mathcal{D}^{d_{max}-1}$ (Second Part of Phase 1) 78
CONTENTS

5.2.3 The MAD calculation for other domain pools (Phase 2) . . 82
5.2.4 Quadtree-based encoding by the results of the MAD calcu-
      lation (Phase 3) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
5.3 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . 84
5.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86

6 Concluding Remarks 88
   6.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
   6.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

A Original Images for Experiment 91

Index 108
List of Figures

2.1 Sierpinski triangles from different initial images. .................. 21
2.2 Fractal image encoding process. ........................................ 27
2.3 PSNR versus the number of decoding steps for $256 \times 256$ Lena en-
coded with the MSE and the MAD criteria. .......................... 28
2.4 PSNR versus decoding steps for both scale and offset calculation
methods................................................................. 31
2.5 First five iterations of the Lena image from the initial Mandrill
image. ................................................................. 33
2.6 Portions of Lena image. (a) Original image enlarged to 4 times.
(b) decoded image....................................................... 34
3.1 An example of the quadtree partitioning. ......................... 37
3.2 Quadtree partitioning for $512 \times 512$ Lena image (1804 blocks,
29.01 PSNR). .......................................................... 38
3.3 Binary-tree partitioning for $512 \times 512$ Lena image (1772 blocks,
29.80 dB PSNR). ..................................................... 39
3.4 An example of alternating binary-tree partitioning (ABP). ...... 40
3.5 Alternating binary-tree partitioning for $512 \times 512$ Lena image (1700 blocks, 29.67 dB PSNR). .................................................. 41
3.6 PSNR versus bit rate for $512 \times 512$ Lena image where the maximum range size from $8 \times 8$ to $64 \times 64$ and the minimum is $4 \times 4$. .......................................................... 43
3.7 Ratio of the number of rectangular blocks to the total range blocks versus the bit rate for Lena, Boat, Peppers, and Girl images. . . . . 44
3.8 Comparison of results for $512 \times 512$ Lena image. ............... 45
3.9 PSNR versus bit rate for $512 \times 512$ Lena, Boat, Peppers, and Girl images using quadtree partitioning and ABP. ................. 46
4.1 The extraction of the half-overlapping domain pool. ............... 50
4.2 The proposed architecture for the first stage in encoding procedure. 52
4.3 The structure of PE of the proposed architecture. ................. 53
4.4 The ring connection of PEs to shift domain blocks to the next PEs. 54
4.5 The structure of the fast comparison module in the case of $C = 4$. 56
4.6 The structure of the fast comparison module in the case of $C = 8$. 57
4.7 An example of rotation operation in the case of $C = 4$. (a) The calculation of the distortion between a range and a domain block. (b) The PPs, denoted by the thick-line, compute the distortion. (c) The current results are shifted along the dotted-line arrow and added to the intermediate results. (d) The same process is repeated for the rotated domain block. .................................................. 58
LIST OF FIGURES

4.8 The connection of the calculation for the range block and the rotated and/or reflected domain blocks. ........................................... 59
4.9 The eight isometric transformed domain blocks of a domain block (D1). ................................................................. 61
4.10 The position and the time of the pixel of D1 that can be rotated to D5. ................................................................. 67
4.11 The modified domain block memory for the larger domain pool. . 68
4.12 The simulation result by Verilog HDL. ............................................. 71
4.13 The Verilog HDL synthesis of FCM with 0.35 μm CMOS technology. 72
4.14 The chip layout of the proposed FCM with 0.35 μm CMOS technology. ................................................................. 74
5.1 The block diagram of PE. ......................................................... 77
5.2 The $d-1$ depth range and domain block consist of four $d$ depth range and domain blocks. (a) range blocks. (b) domain blocks. . 79
5.3 The MAD calculation for other domain pools by the EA modules. 83
5.4 The simulation result of the proposed algorithm for $128 \times 128$ Lena image (26.42 PSNR, 1.29 bpp). ................................. 85
5.5 The simulation result of QPFIC algorithm for $128 \times 128$ Lena image. 86
A.1 $512 \times 512$ Lena image. .................................................. 92
A.2 $512 \times 512$ Boat image. .................................................. 93
A.3 $512 \times 512$ Peppers image. .............................................. 94
A.4 $512 \times 512$ Girl image. .................................................. 95
A.5 $512 \times 512$ Mandrill image. ............................................. 96
List of Tables

2.1 The eight isometric transformations $T_f$. ....................... 26

3.1 Information in bits for the representation of block transformations
in the case of $512 \times 512$ image. .............................. 42

3.2 Performance comparisons. ................................. 47

4.1 The computation steps needed to calculate the distortion about
the eight isometric transformed domain blocks of general pipeline
processing and FCM. ........................................... 60

4.2 The parameters of the eight isometric transformations $M_f$. ... 62

4.3 Estimated encoding time. ................................. 72
# List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>metric space</td>
</tr>
<tr>
<td>$\delta$</td>
<td>metric measure</td>
</tr>
<tr>
<td>$\tau_{i,j}$</td>
<td>contractive transformation</td>
</tr>
<tr>
<td>$\mu_a$</td>
<td>attractor</td>
</tr>
<tr>
<td>$\mu_{\text{orig}}$</td>
<td>original image</td>
</tr>
<tr>
<td>$N \times N$</td>
<td>pixels of gray scale image</td>
</tr>
<tr>
<td>$\mathcal{R}$</td>
<td>range pool</td>
</tr>
<tr>
<td>$r_{i,j}$</td>
<td>$(i,j)^{th}$ range block</td>
</tr>
<tr>
<td>$R \times R$</td>
<td>pixels of a range block</td>
</tr>
<tr>
<td>$N_R \times N_R$</td>
<td>the number of range blocks</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>domain pool</td>
</tr>
<tr>
<td>$d_{m,n}$</td>
<td>$(m,n)^{th}$ domain block</td>
</tr>
<tr>
<td>$d_{\text{tran}}$</td>
<td>spatially transformed domain block</td>
</tr>
<tr>
<td>$D \times D$</td>
<td>pixels of a domain block</td>
</tr>
<tr>
<td>$N_D \times N_D$</td>
<td>the number of domain blocks</td>
</tr>
<tr>
<td>$s_{i,j}$</td>
<td>scaling factor</td>
</tr>
<tr>
<td>$o_{i,j}$</td>
<td>offset value</td>
</tr>
<tr>
<td>$P$</td>
<td>interval parameter</td>
</tr>
<tr>
<td>$T_f$</td>
<td>$f^{th}$ isometric transformation</td>
</tr>
<tr>
<td>$S$</td>
<td>spatial contraction function</td>
</tr>
<tr>
<td>$A(i,j)$</td>
<td>domain block selection function</td>
</tr>
<tr>
<td>$I(i,j)$</td>
<td>isometry selection function</td>
</tr>
<tr>
<td>$b$</td>
<td>the largest possible value of the signal</td>
</tr>
<tr>
<td>$B_{\text{total}}$</td>
<td>the total bits needed to encode a range block</td>
</tr>
<tr>
<td>$N_{\text{tree}}$</td>
<td>the total bits needed to represent an image as a tree</td>
</tr>
<tr>
<td>$N_p \times N_p$</td>
<td>the number of Processing Elements (PEs)</td>
</tr>
<tr>
<td>$E_p$</td>
<td>mapping function to map four range blocks to a contracted domain block by averaging</td>
</tr>
</tbody>
</table>
### List of Tables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \times C$</td>
<td>the number of Pixel Processors (PPs)</td>
</tr>
<tr>
<td>$M_I$</td>
<td>the index mapping function of isometric transformation</td>
</tr>
<tr>
<td>$P_{k,l}(t)$</td>
<td>the position of the pixel $(k,l)$ of a domain block at time $t$</td>
</tr>
<tr>
<td>$(k_d,l_d)$</td>
<td>the base position to calculate the movement of the pixel $(k,l)$ of a</td>
</tr>
<tr>
<td></td>
<td>domain block</td>
</tr>
<tr>
<td>$\mathcal{D}_P$</td>
<td>the the P-pixels-interval domain pool</td>
</tr>
<tr>
<td>$\mathcal{R}^d$</td>
<td>the $d$ depth range pool</td>
</tr>
<tr>
<td>$\mathcal{D}^d$</td>
<td>the $d$ depth domain pool</td>
</tr>
<tr>
<td>$d_{\text{min}}, d_{\text{max}}$</td>
<td>the minimum and maximum depth of domain pool</td>
</tr>
</tbody>
</table>
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABP</td>
<td>Alternating Binary-tree Partitioning</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>CP</td>
<td>Comparator</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>EA</td>
<td>Error Adder</td>
</tr>
<tr>
<td>FCM</td>
<td>Fast Comparison Module</td>
</tr>
<tr>
<td>FIC</td>
<td>Fractal Image Coding</td>
</tr>
<tr>
<td>FID</td>
<td>Fractal Image Decoding</td>
</tr>
<tr>
<td>FPFIC</td>
<td>Fixed-size Partitioning Fractal Image Compression</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>IFS</td>
<td>Iterated Function System</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
</tr>
<tr>
<td>MAD</td>
<td>Mean Absolute Difference</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>PE</td>
<td>Processing Element</td>
</tr>
<tr>
<td>PIFS</td>
<td>Partitioned Iterated Function System</td>
</tr>
<tr>
<td>PP</td>
<td>Pixel Processor</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>QPFIC</td>
<td>Quadtree-based Partitioning Fractal Image Compression</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integrated</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background and Motivations

Digital image processing [1], the manipulation of images by computer, is a relatively recent development in terms of humans’ ancient fascination with visual stimuli. Public awareness of digital image processing has been greatly increased by video games and digital video special effects used in the entertainment industry. The usage of digital image processing in commercial, industrial, and medical applications and in scientific research continues to grow. One can expect digital image processing to play an important role in the future.

Files that are on the Internet are actually stored on computers all over the world. To get the files to your computer they must be transmitted across a network (Web) of phone lines and other cables. The larger the file size, the longer it takes to transfer the data. For this reason web sites try to use files that are small in size. These smaller files take up less space (storage) on the storage devices where they are stored, and take less time to be transported to your computer. Data compression techniques exploit inherent redundancy and irrelevancy by trans-
forming a data file into a smaller file from which the original image file can later be reconstructed, exactly or approximately. Some data compression algorithms are lossless, while others are not. A lossless algorithm eliminates only redundant information, so that one can recover the image exactly upon decompression of the file. A lossy compression algorithm eliminates irrelevant information as well, and thus permits only an approximate reconstruction of the original, rather than an exact duplicate. As one might expect, lossy compression algorithms achieve higher compression ratios.

Digital representations of images usually require a very large number of bits. In many applications, it is important to consider techniques for representing an image, or the information contained in the image, with fewer bits. Since digital images, by their nature, are quite data intensive, reducing their size can produce solutions. By eliminating redundant or unnecessary information, lossy image compression is the activity that addresses this aim. Image compression techniques have been applied in several areas of image and video processing such as TV transmission, video conferencing and portable video telecommunication.

Standard methods of image compression come in several varieties. Currently, the most popular method relies on eliminating high-frequency components of the signal by storing only the low-frequency Fourier coefficients. This method uses a discrete cosine transform (DCT), and is the basis of the so-called JPEG standard. This standard, defined by the Joint Photographic Experts Group, describes ways of taking bit-mapped data for color or gray-scale continuous-tone images and storing it in a smaller number of bytes. Another method, called
vector quantization, breaks up images into a small number of canonical pieces
and storing only a reference to which piece goes where.

Fractal image compression techniques were originally proposed by Barnsley
as a compression method for binary images [3], and applied to gray-scale images
by Jacquin [4], [6]. Fractal coding has received considerable attention because
of its use of the self-similarity in images, an innovation that had not been used
previously in the field of image compression. Many improvements in compression
efficient and decoded image quality have been reported [8]-[19]. These advantages
include [24]:

- **High Compression Ratios.** Fractal image compression can offer higher
  compression ratios than JPEG coding, while maintaining near equivalent
  PSNR values [7].

- **No Codebook.** Fractal image coding used a virtual codebook while en-
  coding the image, and does not require the storage or transmission of any
  part of a codebook or synchronization around a common codebook. Fractal
  image coding defines an image as iterated function systems, with only the
  parameters of this function needing to be stored or transmitted.

- **Fast Decompression.** Fractal image decoding is linear with respect to the
  number of pixels in an image. These iterations will cause the initial image
to converge to the fixed-point solution of the function.

- **Resolution Independence.** The image may be decoded at a different
  resolution than that of the original image, with higher resolution being
obtained through the fractal nature of the image itself.

- **High Quality Decoded Image.** Fractal image coding has been classified as a second generation coding scheme because fractal image compression takes into account the Human Visual System since fractal curves look very similar to natural images.

- **Combination with Other Techniques.** Fractal image coding can easily be adapted to work in combination with other coding techniques.

Despite all of its advantages, fractal image coding is not considered a primary compression scheme due to a long encoding time as opposed to a short decoding time. Moreover, during image compression the encoding time depends on the compression ratio we want to achieve, while during decompression the decoding time is not affected by this value. Even highly optimized fractal image coding algorithms are unable to run in real-time on computers today for any reasonably sized image. Though computationally expensive, the compression algorithm is highly parallelizable, and maps well to parallel architecture.

The first step in the fractal image compression is to partition the image into non-overlapping range blocks. Taken together, the set of range blocks must cover the entire image, but they can be any size or shape. Next, the program defines a collection of possible domain blocks, which must be larger than the range blocks, can overlap, and need not cover the entire image. For each range block, the program must choose the best matching domain block that, after an appropriate affine transformation is applied, most closely matches the range block. This
1.2. PURPOSE OF THIS THESIS

The encoding process generates a file that consists of a header with information about domain blocks and affine coefficients chosen for each range block.

The most computationally intensive operation of fractal image compression is the computation of distances between range and the transformed domain blocks, due to the large amount of combinations between domain and range blocks. To meet high performance, ASICs are required for high-speed fractal image coding.

1.2 Purpose of This Thesis

The first part of this thesis focuses on the development of the partitioning scheme for VLSI design of fractal image coding.

The partitioning schemes are very important for fractal image compression \cite{?}, \cite{18}. There are many methods of partitioning an image based on squares and rectangles such as fixed-size partitioning, quadtree partitioning, and HV partitioning \cite{9}, \cite{?}, \cite{18}. We propose an alternating binary-tree partitioning (ABP) to design a parallel processing architecture for fractal image coding \cite{19}. The ABP scheme repeats the partitioning process to partition rectangular blocks into square blocks or square blocks into rectangular blocks. The ABP scheme is an efficient scheme to implement the dedicated architecture for fractal image coding because it gives better compression and has a regular partitioning scheme.

The second part of this thesis concerns the design for high-speed fractal image compression.

Ancarani et al. \cite{20} designed an ASIC for fractal image compression that performs the eight comparisons between a range block and the transformed domain
blocks in parallel utilizing data independence. To perform the image compression, this architecture needs a host computer providing the chip inputs and storing the output. However, the area cost of this architecture is increased due to the increase of the circuit for parallel processing and the chip must provide all the domain blocks into the chip. A systolic array [27]-[42], proposed by He et al. [21] is capable of computing the multiplication of pixels of range and domain blocks in parallel. This architecture has the external memory in order to store the rotating and/or flipping all the domain blocks. Fatemi et al. [22] proposed a parallel and pipelined architecture called “Fractal Engine”. Affine module in Fractal Engine is capable of executing the eight isometric transformations on a domain block by Reflection Unit and Transposer Unit.

All architectures in literature had the external memory for the storage of domain blocks and utilized global data communication for providing domain blocks to all the processors. As the number of processors increases, expanding non-local communication paths is difficult without slowing down the system clock.

In this thesis, we propose two systolic arrays for high-speed fractal image compression. One is an architecture for fixed-size partitioning fractal image compression (FPFIC) [25]. This architecture is capable of performing the fractal image encoding without the external memory for the fixed domain pool and using only local communication. Each processor has the memory to store a range and a domain block that is shifted to the next processor using local communication. The main features of this architecture are that each processor has a range block and extracts the domain block from the other processors without the exter-
nal memory for the fixed domain pool. This architecture has the fast isometric transformation module, which is capable of performing the comparisons between a range and the eight isometries of a domain block obtained from compositions of reflections and 90 degree rotations by only one full rotation around the center. Since the data needed in the calculation for the reflected domain blocks is obtained by rotating domain blocks, the calculation for all domain blocks is finished during the 360 degree rotation process.

The other architecture, called quadtree-based partitioning fractal image compression (QPFIC), is the modified FPFIC architecture and utilizes quadtree partitioning [26]. The quadtree-based encoding scheme is based on the flexible-size partitioning and yields better performances than fixed-size partitioning encoding schemes. Since large domain blocks consist of small domain blocks, the calculations of distortion for all kinds of domain blocks are performed by using only the maximum-depth domain pool that is extracted from the smallest range blocks of the neighboring processors. This architecture performs the MAD (mean absolute difference) calculation of the maximum-depth domain pool and computes the MADs of other domain pools by adding the MADs of the maximum-depth domain pool. To perform fractal image coding using quadtree partitioning this architecture needs an one time comparison process for the maximum-depth domain pool, while it usually needs to repeat the comparison process for several depth domain pools.
1.3 Organization of This Thesis

This thesis is organized as follows.

Chapter 1 explains the motivations and purposes of this research work. It gives the thesis outline.

Chapter 2 first presents an overview of fractals and fractal image compression. Next, it presents the advantages of fractal image compression, and the problems that are encountered in implementing real-time fractal image compression.

Chapter 3 first presents partitioning schemes of fractal image compression. Then we propose a new partitioning, called alternating binary-tree partitioning (ABP), that is an efficient partitioning scheme for VLSI design of fractal image compression. Finally, experimental results that support the validity of the proposed partitioning scheme are presented.

Chapter 4 proposes an architecture for fixed-size partitioning fractal image compression (FPFIC). Then we present the fast isometric transformation module and analyze it to show the validity of the proposed module. Finally, one Fast Comparison Module (FCM) of this architecture is implemented in Verilog HDL and simulated by Verilog-XL of Cadence company. We show the results of implementing one FCM with the Rohm CMOS 0.35 μm triple metal layer technology.

Chapter 5 proposes an architecture for quadtree-based partitioning fractal image compression (QPFIC). Then we analyze it to show the validity of the proposed architecture. Finally, experimental results by computer simulation, that support the validity of the proposed architecture are presented.

Chapter 6 concludes the concluding remarks and the contributions of this
thesis to the area of image compression. Suggestions for future work are also introduced.
Chapter 2

Fractal Image Compression

2.1 Introduction

A fractal is a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole. Fractals are generally self-similar and independent of scale.

When most people think of fractals, the Mandelbrot and Julia sets immediately spring to mind. These sets are the results of repeatedly iterating a simple mathematical function at different locations (different initial values) until it either converges on a single value, or is found to be unbounded [2].

Another type of fractal of similar form is called an Iterated Function System or IFS. Figure 2.1, called the Sierpinski Triangle, shows several iterations of this process on several input image. What we observe, and what is in fact true, is that all the copies seem to be converging to the same final image. Because the copy process reduces the input image, the copies of any initial image will be reduced to a point as we repeatedly feed the output back as input; there will be more and more copies, but each copy gets smaller and smaller. So, the initial image
Figure 2.1: Sierpinski triangles from different initial images.

doesn’t affect the final image. In fact, it is only the position and the orientation of the copies that determines what the final image will look like.

M. Barnsley and his student A. Jacquin first seriously considered using fractals to compress still images and proposed the Partitioned Iterated Function System (PIFS) [4], [6]. They looked for functions which would generate the desired image with a number of parameters to an IFS. They attempted to break up images to allow several different IFS to work on a single image. Their idea is that instead of using the entire image as the domain, it was possible to partition an image by blocks, independent of the image itself and find an IFS for each block. Each IFS
would map one domain block onto a range block. Jacquin’s work resulted in the
collage theorem, which proved that if each of the mappings is contractive, then an
IFS will converge on a single fixed image after repeated iterations. What makes
this useful is that in natural images there is a remarkable amount of self-similarity
at various scales, and it is generally possible to find a good range-domain mapping
for each block. This allows one to transmit only the IFS mappings and regenerate
the final image without having any other information about the initial image.

Fractal Image Coding (FIC) shows great promise in terms of high compression
ratios and better quality at high compression ratios, relative to other methods.

2.2 Theoretical Foundations

In this section, we briefly present the fundamental principles of fractal image
coding based on a theory of contractive iterated image transformations [9]. IFS’s
served as the motivating concept behind the development of fractal image com-
pression, and most of the work on fractal image compression is based on IFS or
its generalization.

**Definition 2.1 (Contractive Maps and IFS’s)** Let M be a metric space with
metric δ. A map \( \tau : M \to M \) is Lipschitz if \( \tau \) satisfies that there exists a positive
real value \( s \), such that

\[
\delta(\tau(\mu), \tau(\nu)) \leq s\delta(\mu, \nu), \quad \forall \mu, \nu \in M.
\]  

(2.1)

The value \( s \) is called Lipschitz factor. If the Lipschitz factor satisfies \( s < 1 \), then
\( \tau \) is said to be contractive with contractivity factor \( s \).
2.2. THEORETICAL FOUNDATIONS

The inverse problem of iterated transformation theory is the construction of a contractive image transformation \( \tau \), which is contractive on a metric space \((M, \delta)\).

**Definition 2.2 (Cauchy Sequence)** A sequence of points \( \{\mu_n\} \) in a metric space is called a Cauchy sequence if for any \( \epsilon > 0 \), there exists an integer \( N \) such that

\[
\delta(\mu_m, \mu_n) < \epsilon, \text{ for all } n, m > N. \tag{2.2}
\]

**Theorem 2.1 (The Contractive Mapping Fixed-Point Theorem)** Let \( M \) be a complete metric space and \( \tau : M \to M \) be a contractive mapping. Then there exists a unique point \( \mu_a \in M \) such that for any point \( \mu \in M \)

\[
\mu_a = \tau(\mu_a) = \lim_{n \to \infty} \tau^n(\mu), \quad \mu_a \in M. \tag{2.3}
\]

Such a point is called a fixed point or the attractor of the mapping \( \tau \).

**Proof:** Select \( \mu \in M \). Then for \( n > m \),

\[
\delta(\tau^m(\mu), \tau^n(\mu)) < s^n\delta(\tau^{m-1}(\mu), \tau^{n-1}(\mu)) < s^m\delta(\mu, \tau^{n-m}(\mu)). \tag{2.4}
\]

Now, using this and the triangle inequality repeatedly,

\[
d(\mu, \tau^k(\mu)) \leq \delta(\mu, \tau^{k-1}(\mu)) + \delta(\tau^{k-1}(\mu), \tau^k(\mu)) \\
\leq \delta(\mu, \tau(\mu)) + \delta(\tau(\mu), \tau(\tau(\mu))) + \cdots + \delta(\tau^{k-1}(\mu), \tau^k(\mu)) \\
\leq (1 + s + \cdots + s^k + s^k)\delta(\mu, \tau(\mu)) \\
\leq \frac{1}{1 - s}\delta(\mu, \tau(\mu)). \tag{2.5}
\]
2.3. Fractal Image Coding Algorithm

We can then write Equation (2.4) as

\[ \delta(\tau^n(\mu), \tau^m(\mu)) \leq \frac{s^m}{1-s} \delta(\mu, \tau(\mu)), \] (2.6)

and since \( s < 1 \), the left side can be made as small as we like if \( n \) and \( m \) are sufficiently large. This means that the sequence \( \mu, \tau(\mu), \tau(\tau(\mu)), \ldots \) is a Cauchy sequence, and since \( M \) is complete, the limit point \( \mu_a = \lim_{n \to \infty} \tau^n(\mu) \) is in \( M \). \( \tau \) is continuous and \( \tau(\mu_a) = \tau(\lim \tau^n(\mu)) = \lim \tau^{n+1}(\mu) = \mu_a \).

**Colloquary 2.1 (Collage Theorem)** With the hypothesis of the Contractive Mapping Fixed Point Theorem,

\[ \delta(\mu, \mu_a) \leq \frac{1}{1-s} \delta(\mu, \tau(\mu)). \] (2.7)

**Proof:** This is a consequence of taking the limit as \( k \) goes to infinity in Equation (2.5).

### 2.3 Fractal Image Coding Algorithm

Color images are typically extensions of the gray scale representation of an image, so we will focus on gray scale models. A gray scale image is represented as a collection of discrete picture elements or pixels. Each pixel value takes on discrete values (typically ranging from 0 to 255) representing a gray level.

Provided that \( \mu_{\text{orig}} \) is an \( N \times N \) pixel gray scale image, we partition the original image \( \mu_{\text{orig}} \) into a set of non-overlapping \( R \times R \) pixel range blocks \( \{ r_{i,j} \} \), as follows:

\[ \mu_{\text{orig}} = \bigcup_{i,j=1}^{N_R} r_{i,j}, \quad r_{i,j} \cap r_{i',j'} = \emptyset \quad \text{for} \quad (i,j) \neq (i',j'), \] (2.8)
where $r_{i,j}$ represents the range block at coordinates $(i, j)$, and $N_R \times N_R$ is the total number of range blocks. $\mathcal{R} = \{r_{i,j}|1 \leq i, j \leq N_R\}$ is called the range pool. A set of overlapping $D \times D$ ($D = 2R$) pixel domain blocks, $\{d_{m,n}|1 \leq m, n \leq N_D\}$, are drawn from the domain pool $\mathcal{D}$ where $N_D \times N_D$ is the total number of domain blocks.

A variety of domains are used in the literature. Because fixed-spacing domain pool has a regular data flow, it is very efficient to implement VLSI architecture. The domain pool we will use in our main design is the half-overlapping domain pool $\mathcal{D}$ which is overlapped along the vertical and horizontal directions with the overlapping interval parameter $P$ set to $D/2$ pixels and $N_D = N_R - 1$.

For every range block, the best matching domain block is searched for among all domain blocks by performing a set of transformations on the blocks. Generally speaking, there are two searching techniques. One is full-search and the other is fast-search [21]. The full-search algorithm compares each range block with all the domain blocks while a fast-search algorithm compares each range block with some particular domain blocks. The full-search algorithm can produce better performance than fast-search algorithms and possesses a very regular data flow. However, with the fixed range block size, even the full-search algorithm alone yields worse performance than flexible-block-size algorithm. Considering the trade-offs among the degrees of performance and VLSI implementation, fixed-block-size full-search fractal image coding is adopted in this chapter.

The mapping for the $(i, j)^{th}$ range block, $\tau_{i,j}$, consists of a scaling factor $s_{i,j}$, offset $o_{i,j}$, isometric transformation $T_f$, $1 \leq f \leq 8$, and spatial contraction $S$. 
Table 2.1: The eight isometric transformations $T_f$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identity</td>
</tr>
<tr>
<td>2</td>
<td>Rotation through $+90^\circ$</td>
</tr>
<tr>
<td>3</td>
<td>Rotation through $+180^\circ$</td>
</tr>
<tr>
<td>4</td>
<td>Rotation through $+270^\circ$</td>
</tr>
<tr>
<td>5</td>
<td>Reflection about mid-vertical axis</td>
</tr>
<tr>
<td>6</td>
<td>Reflection and Rotation $-90^\circ$</td>
</tr>
<tr>
<td>7</td>
<td>Reflection and Rotation $-180^\circ$</td>
</tr>
<tr>
<td>8</td>
<td>Reflection and Rotation $-270^\circ$</td>
</tr>
</tbody>
</table>

An isometric transformation $T_f$ maps a square block to one of eight isometries obtained from compositions of reflections and 90 degree rotations as shown in Table 2.1.

The result of applying this mapping is an approximation to the $(i, j)^{th}$ range block, $\tilde{r}_{i,j}$ consists of a contrast scaling $s_{i,j}$, offset $\alpha_{i,j}$ pixel shuffling, isometric transformation $T_f : \mathcal{R} \rightarrow \mathcal{R}$ and spatial contraction function $S : \mathcal{D} \rightarrow \mathcal{R}$ as follows:

$$
\tilde{r}_{i,j} = s_{i,j}T_f S(d_{A(i,j)}) + \alpha_{i,j},
$$

(2.9)

where $A(i,j)$ is a domain block selection function which associates the $(i,j)^{th}$ range block with a domain block from $\mathcal{D}$, and $I(i,j)$ is an isometry selection function which maps the $(i,j)^{th}$ range block to one of possible isometric transformations.

Fractal Image Coding (FIC) process consists of determining, for all range blocks, the mapping parameters in Equation (2.9) such that the distortion between each range block and its approximation, $\delta(\tilde{r}_{i,j}, r_{i,j})$, is minimized as shown....
in Figure 2.2.

Common distortion criteria include the mean square error (MSE) and mean absolute difference (MAD) as Equations (2.10) and (2.11), respectively.

\[
\delta_{\text{MSE}}(\tilde{r}_{i,j}, r_{i,j}) = \sqrt{\sum_{k=1}^{R} \sum_{l=1}^{R} (\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l))^2}
\]

\[
= \sqrt{\sum_{k,l=1}^{R} (s_{i,j}d_{\text{tran}}(k,l) + \alpha_{i,j} - r_{i,j}(k,l))^2}, \quad (2.10)
\]
2.3. FRACTAL IMAGE CODING ALGORITHM

![Graph showing PSNR versus the number of decoding steps for 256x256 Lena encoded with the MSE and the MAD criteria.](image)

Figure 2.3: PSNR versus the number of decoding steps for 256×256 Lena encoded with the MSE and the MAD criteria.

\[
\delta_{\text{MAD}}(\tilde{r}_{i,j}, r_{i,j}) = \sum_{k=1}^{R} \sum_{l=1}^{R} |\tilde{r}_{i,j}(k, l) - r_{i,j}(k, l)|
\]

\[
= \sum_{k,l=1}^{R} |s_{i,j}d_{\text{tran}}(k, l) + a_{i,j} - r_{i,j}(k, l)|,
\]

(2.11)

where \(d_{\text{tran}}(k, l) = T_{ij} S(d_{A(i,j)}(k, l))\) and \(\sum_{k,l=1}^{R} \overset{\text{def}}{=} \sum_{k=1}^{R} \sum_{l=1}^{R}\).

In practice, the Peak Signal-to-Noise Ratio (PSNR) is used to measure the difference between two images. It is defined as
\[ PSNR = 20 \log_{10} \left( \frac{b^2}{\delta(\delta, \delta)} \right) , \]  

(2.12)

where \( b \) is the largest possible value of the signal (typically 255). The PSNR is given in decibel units (dB) which measure the ratio of the peak signal to the difference between two images.

Figure 2.3 shows the PSNR versus the number of decoding steps for 256×256 Lena encoded with the MSE and the MAD criteria. The results allow us to conclude that the MAD criterion does not affect the quality of the decoded images. In order to simplify the architecture and to increase the performance speed, the distortion criterion adopted is the MAD criterion in this thesis.

MSE metric allows easy computation of optimal values for \( s_{i,j} \) and \( a_{i,j} \) in Equation (2.9). This will give us contrast and brightness settings that make the linearly transformed \( d_{\text{tran}} \) values have the least squared distance from the \( r_{i,j} \) values. The minimum of \( \delta_{MSE}^2 \) occurs when the partial derivatives with respect to \( s_{i,j} \) and \( a_{i,j} \) are zero, which occurs when

\[
s_{i,j} = \frac{R^2 \sum_{k,l=1}^{R} r_{i,j}(k,l) d_{\text{tran}}(k,l) - \sum_{k,l=1}^{R} r_{i,j}(k,l) \sum_{k,l=1}^{R} d_{\text{tran}}(k,l)}{R^2 \sum_{k,l=1}^{R} d_{\text{tran}}^2(k,l) - \left( \sum_{k,l=1}^{R} d_{\text{tran}}(k,l) \right)^2} \tag{2.13}
\]

and

\[
a_{i,j} = \frac{1}{R^2} \left( \sum_{k,l=1}^{R} r_{i,j}(k,l) - s_{i,j} \sum_{k,l=1}^{R} d_{\text{tran}}(k,l) \right) . \tag{2.14}
\]
2.3. FRAC TAL IMAGE CODING ALGORITHM

Ida et al. [5] set offset value \( a_{i,j} \) as the average value of the range block and distortion criterion as the distortion between range and domain blocks that subtract their average values, respectively:

\[
\delta_{MAD}(\hat{r}_{i,j}, r_{i,j}) = \sum_{k,l=1}^{R} |s_{i,j} \hat{d}_{tran}(k,l) + a_{i,j} - r_{i,j}(k,l)|
\]

\[
= \sum_{k,l=1}^{R} |s_{i,j} \hat{d}_{tran}(k,l) - \hat{r}_{i,j}(k,l)|,
\]  

(2.15)

\[
s_{i,j} = \frac{\sum_{k,l=1}^{R} \hat{r}_{i,j}(k,l) \hat{d}_{tran}(k,l)}{\sum_{k,l=1}^{R} \hat{d}_{tran}^2(k,l)},
\]  

(2.16)

\[
a_{i,j} = \frac{1}{R^2} \sum_{k,l=1}^{R} r_{i,j}(k,l),
\]  

(2.17)

where \( \hat{r}_{i,j} = r_{i,j} - \frac{1}{R^2} \sum r_{i,j} \) and \( \hat{d}_{tran} = d_{tran} - \frac{1}{R} \sum d_{tran} \).

Figure 2.4 shows the PSNR as a function of the number of decoding steps using the general scale and offset calculation (Equations (2.13), (2.14)) and Ida’s method (Equations (2.16), (2.17)), respectively for the 512 \( \times \) 512 Lena image. Since the decoding process of Ida’s method starts from the average of range blocks, the curve of the new calculation converges in fewer iterations. We will use Equations (2.16) and (2.17) in this thesis to reduce the complexity of scale and offset calculation and the six iteration of decoding process by using Ida’s method.
Figure 2.4: PSNR versus decoding steps for both scale and offset calculation methods.

2.4 Fractal Image Decoding Algorithm

Fractal Image Decoding (FID) consists of iterating $\tau$ from any initial image. We partition the initial image into domain blocks specified in the code file. For each domain block, we read the coefficients of isometric transformation from the code file and map this domain block into range region. The domain pixel values are multiplied by $s_{i,j}$ and are added $o_{i,j}$, and placed in the location in the range blocks determined by $T_f$ information. This constitutes of one decoding iteration. The decoding step is iterated until the fixed point is approximated; that is, until further iteration doesn’t change the image or until the change is below some small
2.4. *Fractal Image Decoding Algorithm*

threshold iteration value. Figure 2.5 shows the first five iterations of a decoding sequence corresponding to a fractal block code for the $512 \times 512$ Lena image with the initial Mandrill image.

One of advantages of FIC is resolution independence. This means that the image may be decoded at a different resolution than that the original image, with higher resolution being obtained through the fractal nature of the image itself. Therefore, when we decode, we may select any resolution we wish. Figure 2.6 shows portions of both the original Lena image and a version generated by decoding an encoding of Lena at 4 times the original size. For comparison, the original version is shown, magnified by a factor of 4. It is known that the decoded image has better resolution than the simply magnified image.
Figure 2.5: First five iterations of the Lena image from the initial Mandrill image.
Figure 2.6: Portions of Lena image. (a) Original image enlarged to 4 times. (b) decoded image.
Chapter 3

Alternating Binary-tree Partitioning (ABP) for Fractal Image Coding

3.1 Introduction

The first step of FIC is to partition the original image into range blocks and domain blocks. The partitioning scheme of the encoding process is very important for fractal image compression [?], [18] because reducing the total number of range blocks is increasing of the compression of the image. VLSI Architectures proposed in literature are based on fixed-size partitioning or quadtree partitioning to partition the original image into range blocks because their schemes are very regular and suitable for VLSI implementation. In this chapter, we propose an efficient partitioning scheme, called alternating binary-tree partitioning (ABP) that is based on binary-tree partitioning and yields better performances than fixed-size or quadtree partitionings. Therefore, ABP is an efficient partitioning scheme for the special-purpose hardware for high-speed fractal image coding because its data flow is regular. In Section 3.5, we show the simulation results of
3.2 Conventional Partitioning Schemes of FIC

Fisher [9] introduced adaptive partitionings such as quadtree partitioning and HV partitioning in order to increase the decoded image quality of fractal methods. Fixed-size partitioning and quadtree partitioning are partitioning schemes based on squares, while HV partitioning is based on rectangles.

A fixed-size partitioning is that the sizes of range blocks are the same and the best matching domain block corresponding to each fixed-size range block is searched for among all domain blocks by performing a set of transformations on the blocks. Therefore, the encoding algorithm with fixed-size partitioning is very simple. However, there are many regions of the image that are difficult to cover well this way for a given range size. To improve the performance of the coding, we have to reduce the total number of range blocks.

A quadtree partitioning [9] is a representation of an image as a tree in which each node, corresponding to a square portion of the image, contains four sub-nodes, corresponding to the four quadrants of the square (see Figure 3.1. The root of the tree is the initial image. In quadtree partitioning, a square in the image is broken up into four equal-sized sub-squares when it is not covered well enough by a domain block from domain pool $\mathcal{D}$. All the potential domain blocks are compared with each range block. If the distortion value between a range block and the best domain block is above a threshold value and the depth of the quadtree is less than a maximum depth, then the range block is divided into
Figure 3.1: An example of the quadtree partitioning.

four sub-blocks, and this process is repeated. If the distortion value is below a threshold value, the information of the best domain block and its transformation is stored. The image will be encoded by storing each range block as the parameters in Equation (2.9) and a quadtree level $d$. The partitioning result for $512 \times 512$ Lena image is shown in Figure 3.2, where the maximum range size is $32 \times 32$ and the minimum is $4 \times 4$.

HV partitioning is more flexible and uses an enormous domain pool $D$. A rectangle is recursively partitioned either horizontally or vertically to form two new rectangles. The reconstructed images show high qualities but the computational requirements for this scheme are quite large due to the huge domain pool. The encoding process of HV partitioning possesses a very complicated data flow [9]. It is difficult to design an architecture for high-speed fractal image compression by HV partitioning.

Recently, several architectural prototypes for fractal image and video com-
Figure 3.2: Quadtree partitioning for $512 \times 512$ Lena image (1804 blocks, 29.01 PSNR).

...pression have been proposed [20]-[24]. All of them used fixed-size partitioning or quadtree partitioning to partition the original image into range blocks.

### 3.3 Binary-tree Partitioning

A binary-tree partitioning represents an image as a tree in which each node has two sub-nodes while each node has four sub-nodes in quadtree partitioning. Each node denotes a further recursing or ensuing transformation information. A block
is recursively partitioned either horizontally or vertically but the partitioning positions are fixed on the middle of horizontal or vertical lines. Figure 3.3 shows the partitioning result for $512 \times 512$ Lena image by binary-tree partitioning. The image is partitioned into a smaller number of range blocks than quadtree partitioning ($1804$ blocks, $29.01$ dB PSNR). However, the bits of each node are increased due to two partitioning directions (horizontal or vertical directions).
3.4 Alternating Binary-tree Partitioning (ABP)

We propose an alternating binary-tree partitioning (ABP) to partition the image into range blocks $\mathcal{R}$. Figure 3.4 shows an example of ABP. First, the original image is partitioned into square blocks. Then we repeat the partitioning square block into two rectangle blocks and rectangle block into two square blocks if $\delta(r, \bar{r})$ is above a threshold value and the depth of the binary-tree is less than a maximum depth. The positions of the partitioning square block into rectangular blocks are the middle points of horizontal or vertical lines but the positions of the partitioning a rectangular block into square blocks are fixed.

Since the partitioning positions are restricted and the partitioning from rect-
3.4. **ALTERNATING BINARY-TREE PARTITIONING (ABP)**

![Binary Tree Diagram]

Figure 3.5: Alternating binary-tree partitioning for $512 \times 512$ Lena image (1700 blocks, 29.67 dB PSNR).

angles into squares can reduce the bits of node such as quadtree partitioning, ABP gives better performances than binary-tree partitioning.

Figure 3.5 shows that the number of the blocks partitioned by ABP is smaller than binary-tree partitioning.
Table 3.1: Information in bits for the representation of block transformations in the case of $512 \times 512$ image.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaling factor ($s_i$)</td>
<td>5</td>
</tr>
<tr>
<td>offset ($o_i$)</td>
<td>7</td>
</tr>
<tr>
<td>pixel isometry ($t_k$)</td>
<td>3</td>
</tr>
<tr>
<td>position of domain block</td>
<td>14</td>
</tr>
</tbody>
</table>

### 3.5 Experimental Results

The test images are the $512 \times 512$, 8-bpp Lena, Boat, Peppers, and Girl images. In order to compute a bit rate, we compute the information in bits needed to represent a block transformation given in Table 3.1.

The resulting bit rate is

$$\frac{B_{total}N R + B_{tree}}{N^2} \text{[bits/pixel]},$$

where $B_{total}$ and $B_{tree}$ are the total bits needed to encode a range block and to represent an image as a tree, respectively.

Partitionings are computed by letting threshold values of distortion take on the several values. The results are summarizing in Figures 3.6 to 3.9.

Figure 3.6 shows the results encoded changing the maximum size from $8 \times 8$ to $64 \times 64$ for $512 \times 512$ Lena image. This result implies the best maximum size is $16 \times 16$. Let the largest and smallest block sizes be $16 \times 16$ and $4 \times 4$, respectively in this thesis.

Figure 3.7 shows that the ratio of the number of rectangular blocks to the total range blocks is high from the bit rate 0.2 to 0.4 and the effectiveness of ABP
3.5. **EXPERIMENTAL RESULTS**

Figure 3.6: PSNR versus bit rate for $512 \times 512$ Lena image where the maximum range size from $8 \times 8$ to $64 \times 64$ and the minimum is $4 \times 4$.

seems be caused by this fact. Figure 3.8 shows PSNR versus the bit rate results for $512 \times 512$ Lena image using quadtree partitioning, binary-tree partitioning, and ABP and implies ABP results are the best results and binary-tree is better than quadtree partitioning. Figure 3.9 shows that ABP yields lower bit rates and higher qualities than quadtree partitioning for all test images.

Table 3.2 shows the performance comparison between the implementations of quadtree partitioning, binary-tree partitioning, and ABP for $512 \times 512$ Lena image on a 333 MHz Pentium II platform. ABP is slower than quadtree partitioning.
3.5. EXPERIMENTAL RESULTS

Figure 3.7: Ratio of the number of rectangular blocks to the total range blocks versus the bit rate for Lena, Boat, Peppers, and Girl images.

running on the same host PC because it needs to decide the dividing direction to divide two blocks on ABP scheme.

A good architecture for VLSI implementation should possess the property such that the data flow is simple and regular. ABP scheme is more flexible than quadtree partitioning and more regular than HV since the positions of the partitioning are limited. Therefore, ABP is an efficient partitioning scheme for the special-purpose hardware for high-speed fractal image coding because its data flow is regular and ABP will produce good results for any image.
Figure 3.8: Comparison of results for $512 \times 512$ Lena image.

3.6 Summary

A new partitioning scheme called ABP (Alternating Binary-tree Partitioning) for FIC that is suitable for VLSI architecture has been proposed. The partitioning algorithm in the proposed scheme is based on rectangular partitionings such as fixed-size partitioning, quadtree partitioning and HV partitioning. The key idea in the proposed partitioning is to reduce the number of ranges by using rectangles and fix the positions of the partitioning to decrease bits needed to represent the positions of domain blocks. ABP scheme is more flexible than quadtree partition-
Figure 3.9: PSNR versus bit rate for $512 \times 512$ Lena, Boat, Peppers, and Girl images using quadtree partitioning and ABP.

...ing and more regular than HV since the positions of the partitioning are limited. ABP yields lower bit rates and higher qualities than quadtree partitioning for all test images. ABP is an efficient partitioning scheme for the special-purpose hardware for FIC due that its data flow is regular and ABP will produce good results for any image.
Table 3.2: Performance comparisons.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>256×256</td>
<td>91.3</td>
<td>305.9</td>
<td>190.0</td>
</tr>
<tr>
<td>512×512</td>
<td>585.3</td>
<td>2012.5</td>
<td>1216.3</td>
</tr>
</tbody>
</table>
Chapter 4

An Architecture for Fixed-Size Partitioning Fractal Image Compression

4.1 Introduction

The main problem of fractal image coding (FIC) is the tremendous encoding time needed due to the large amount of comparisons between range and domain blocks. To overcome this problem, a few dedicated architectures proposed have utilized global data communication for providing domain blocks to all the processors [20]-[24]. As the number of processors increases, expanding non-local communication paths is difficult without slowing down the system clock. In this chapter, we propose an efficient VLSI architecture for fixed-size partitioning fractal image compression (FPFIC), which uses only local communication. The main feature of this architecture is that it is capable of performing the fractal image encoding without the external memory for the fixed domain blocks. We propose the fast comparison module (FCM) in each processor, which can calculate the eight isometric transformations in one full rotation around the center. This mod-
ule is capable of computing the distortions between the reflected domains and range blocks using the calculation for the rotated domains by the data dependence between domains. This module could be used in many image processing applications which need the isometric transformations. In Section 4.2.4 of this chapter, we show the validity of the operations of FCM. In Section 4.4, FCM is implemented and laid out in 0.35 \( \mu m \) CMOS technology. The FCM chip contains 334252 transistors in a die size of 2.9 \( \times \) 2.9 mm\(^2\). Simulation shows that it has the capability to run up to 40 MHz, or 25 ns per cycle. Finally, we describe the architecture for the larger domain pool.

### 4.2 The Proposed FPFIC architecture

The proposed FPFIC architecture is a systolic array that is arranged in two-dimensional space by the same processor elements (PEs) that are locally connected and are capable of computing range-domain distortions. Each range block pre-loaded to each PE is compared with a single domain block in parallel. After the comparison process, all the domain blocks are shifted and the process is repeated.

Fast-search is used in sequential fractal image encoding to reduce the number of range-domain comparisons but it was not chosen in this thesis because it would require random movements of range or domain blocks across the processor array and is difficult to implement this architecture by VLSI technology.

The use of the large domain pool enables more accurate quantization of subtrees of the image, but it results in an increase of the cost of storing the quantization parameters and computational complexity. In the design of FPFIC, the
half-overlapping domain pool is considered at first.

Assume that there are $N_p \times N_p$ PEs. Since we assume the total number of PEs is equal to the number of range blocks, $N_p = N_R$. The total number of domain blocks is $(N_p - 1)^2$ because the overlapping interval is $D/2 = R$. Notice that the size of the domain pool $\mathcal{D}$ is smaller than the size of the range pool $\mathcal{R}$. 
4.2. **THE PROPOSED FPfIC ARCHITECTURE**

Figure 4.1 shows an example of the extracting domain pool in the case of \( N_R = 4 \). In general, a domain block \( d_{i,j} \) is drawn from four neighboring range blocks. To save memory, the domain blocks are contracted in advance and are stored in PEs. Let \( d_{i,j} \) be a contracted domain block instead of the original domain block. The domain block \( d_{i,j} \) is determined as follows:

\[
d_{i,j} = E_d(r_{i,j}, r_{i+1,j}, r_{i,j+1}, r_{i+1,j+1}),
\]  

where \( E_d \) operator maps four range blocks to a contracted domain block by averaging (or sub-sampling). The size of the domain block is the same as the range block. The total number of domain blocks extracted from the range pool is \( (4 - 1) \times (4 - 1) = 9 \).

We assume that each range block is loaded into each PE. The encoding procedure of the proposed architecture consists of two stages.

### 4.2.1 The Extraction of the Domain Pool : Stage I

In the first stage, the domain pool is extracted from the range blocks. Figure 4.2 shows the connection of PEs during the first stage. The directions of the data flow in each PE are determined by data dependence obtained from Equation (4.1).

The structure of PE of the proposed architecture is shown in Figure 4.3. Each PE has three memory modules, fast comparison module (FCM), contractive mapping module, 1-to-2 decoder, and 2-to-1 multiplexer. When CTRL = 0 occurs, PEs operate the first stage. However, when CTRL = 1 occurs, they operate the second stage. Three memory modules consist of range block memory, domain
4.2. THE PROPOSED FPIC ARCHITECTURE

Figure 4.2: The proposed architecture for the first stage in encoding procedure.

block memory, and code memory which is used to store the results obtained. Each range block is loaded into each range block memory.

In the literature, the architectures have special memory modules to store the domain pool and memory bandwidth becomes a bottleneck, since all PEs access the same memory to receive domain blocks. In this thesis, we propose a systolic array which resolves this problem by using only local data communication.

The contractive mapping module maps a range block to the quarter-size domain block by averaging (or sub-sampling), and then sends the quarter-size domain block to its three neighboring PEs and domain block memory of itself.
4.2. THE PROPOSED FPFIC ARCHITECTURE

![Diagram](image)

Figure 4.3: The structure of PE of the proposed architecture.

4.2.2 Determining the Best Domain Block: Stage II

The second stage determines the best domain block of each range block. The total number of comparisons between the range pool and the domain pool is $(N_R)^2 \times (N_D)^2 \times 8$. In this architecture, each range block is compared with a single domain block in parallel and all domain blocks are shifted to the next PEs. Each PE has a range block and computes the distortion without shifting a domain block to the other PEs by ring connection as shown in Figure 4.4. $(N_R)^2 \times 8$ steps are needed to search the best matching domain block, where there are eight isometric transformations.
4.2. THE PROPOSED FPFC ARCHITECTURE

Figure 4.4: The ring connection of PEs to shift domain blocks to the next PEs.

4.2.3 The Fast Comparison Module (FCM)

The data independence permits the computation of eight comparisons between the transformed domain blocks and one range block in parallel by means of the dedicated hardware architecture. However, the cost of area is significantly increased due to the increase of domain blocks and the circuit for parallel processing. In this section, we propose an efficient architecture for the eight isometric transformations without the external memory for domain blocks. To implement fast isometric transformations, this architecture performs fast rotation of domain block using shifting to the next cell without buffers needed to load and draw out another block. The fast comparison algorithm is:

```c
Fast Comparison Algorithm(R,D1);
{
    While (degrees of rotation is less than 360) {
```
If (degrees is 0/90/180/270)

Calculate the distortion (R, D1/D2/D3/D4);
Calculate and send the intermediate results
of the reflected domains (D5-8);

Rotate the current domain around center of block;
}

Compare 8 results for the domains (D1-8) and select the best;
}

We first describe the fast comparison module (FCM) which is capable of executing the eight isometric transformations on domain blocks and selecting the best transformation among them. Then, in Section 4.2.4, we show the validity of the operations.

This module performs the eight isometric transformations in one full rotation around the center. The proposed architecture consists of \(C \times C\) pixel processors (PPs) to calculate the distortion between two blocks, the eight buffers and CPs to compare the distortion results. Figures 4.5 and 4.6 show the architectures of FCM in the case of \(C = R = 4\) and 8, respectively.

The unit delay factors are utilized to create a proper movement of data. Each PP in FCM has one pixel of the range block and domain block. The data of domain block is shifted to the next PP along the solid-line arrow in each data access clock cycle. Each PP calculates the absolute difference between the two pixels, and sends the intermediate result to the next PP along the dotted-line arrow.
Figure 4.5: The structure of the fast comparison module in the case of \( C = 4 \).

Figure 4.7 shows an example of rotation operation in the case of \( C = 4 \). The distortion calculation between a range block and a domain block is performed (see Figure 4.7(a)). Each column of PPs computes the sum of pixel differences and adds the sum to the input from the left column of PPs. The results are shifted to the right column of PPs simultaneously. The right-most column of PPs sends the sums of pixel differences from top to bottom. \( 2C \) steps are needed for this calculation. The total sum of pixel differences is fed to the four buffers which are used to store the distortion results for the four rotated domain blocks.

The results of the distortion calculation for the reflected domain blocks are obtained during the rotation process. The calculation for the reflected domains can be performed on diagonal PPs, denoted by the thick-lines, during the dis-
Figure 4.6: The structure of the fast comparison module in the case of $C = 8$.

tortion calculation (see Figure 4.7(b)). After one step rotation, the intermediate sums of the distortion for the reflected domain blocks are added to the distortion for itself and are shifted to the next PPs (see Figure 4.7(c)). When the degree
Figure 4.7: An example of rotation operation in the case of $C = 4$. (a) The calculation of the distortion between a range and a domain block. (b) The PPs, denoted by the thick-line, compute the distortion. (c) The current results are shifted along the dotted-line arrow and added to the intermediate results. (d) The same process is repeated for the rotated domain block.
of the rotation is 90, the calculation of the absolute difference between a range block and the rotated domain block is performed (see Figure 4.7(d)).

When four rotation processes are finished, the sums of pixel differences of the four reflected domain blocks are stored in eight PPs denoted by thick lines in Figure 4.5 (or Figure 4.8), and are fed into four buffers, respectively as shown in Figure 4.8. To obtain the results of the four reflected domain blocks, 2C steps

Figure 4.8: The connection of the calculation for the range block and the rotated and/or reflected domain blocks.
4.2. THE PROPOSED FPFIC ARCHITECTURE

Table 4.1: The computation steps needed to calculate the distortion about the eight isometric transformed domain blocks of general pipeline processing and FCM.

<table>
<thead>
<tr>
<th></th>
<th>Pipeline Processing</th>
<th>FCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loading D1 in the array</td>
<td>$C$</td>
<td>$C$</td>
</tr>
<tr>
<td>Rotated domains (D1-4):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loading in and out (D2-4)</td>
<td>$3 \times C$</td>
<td>$3 \times (C/2)$</td>
</tr>
<tr>
<td>Distortion calculation</td>
<td>$4 \times 2C$</td>
<td>$4 \times 2C$</td>
</tr>
<tr>
<td>Reflected domains (D3-8):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loading in and out</td>
<td>$4 \times C$</td>
<td>$0$</td>
</tr>
<tr>
<td>Distortion calculation</td>
<td>$4 \times 2C$</td>
<td>$0$</td>
</tr>
<tr>
<td>Adding the intermediate results</td>
<td>$0$</td>
<td>$2C$</td>
</tr>
<tr>
<td>Total steps</td>
<td>$24C$</td>
<td>$12C + C/2$</td>
</tr>
</tbody>
</table>

are needed.

Table 4.1 compares the computation steps needed to calculate the distortion about the eight isometric transformed domain blocks of general pipeline processing and FCM. First, the domain block (D1) shifted from neighboring PE is loaded in parallel into PPs. $C$ steps are usually needed to exchange one domain block loaded in PPs to the next domain block in pipeline processing. However, $C/2$ steps are needed for a counterclockwise rotation of 90 degrees in this module. The total steps needed to perform four 90 degree rotations of a domain block is $4 \times (C/2)$. Total steps of general pipeline processing and FCM are $24C$ and $12C + C/2$, respectively. It comes out that by using FCM gains a speed-up of 2 times over the general pipeline processing.

The eight results within the buffers are compared to each other using comparators (CPs) to find the smallest distortion value. The number of steps required
### 4.2. THE PROPOSED FPFIC ARCHITECTURE

<table>
<thead>
<tr>
<th>Identity</th>
<th>Rotation +90</th>
<th>Rotation +180</th>
<th>Rotation +270</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1 (f=1)</td>
<td>D2 (f=2)</td>
<td>D3 (f=3)</td>
<td>D4 (f=4)</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>l</td>
<td>l</td>
<td>l</td>
<td>l</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reflection about mid-vertical axis</th>
<th>Reflection and Rotation -90</th>
<th>Reflection and Rotation -180</th>
<th>Reflection and Rotation -270</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5 (f=5)</td>
<td>D6 (f=6)</td>
<td>D7 (f=7)</td>
<td>D8 (f=8)</td>
</tr>
<tr>
<td>k</td>
<td>k</td>
<td>k</td>
<td>k</td>
</tr>
<tr>
<td>-3.3, -1.3, 1.3, 3.3</td>
<td>3.3, 3.1, 3.1, 3.3</td>
<td>3.3, 1.3, -1.3, -3.3</td>
<td>-3.3, 3.1, -1.3, -3.3</td>
</tr>
<tr>
<td>-3.1, -1.1, 1.1, 3.1</td>
<td>1.3, 1.1, 1.1, 1.3</td>
<td>3.1, 1.1, -1.1, -1.3</td>
<td>-1.3, 1.1, -1.1, -1.3</td>
</tr>
<tr>
<td>-3.1, -1.1, 1.1, 3.1</td>
<td>-1.3, -1.1, -1.1, -1.3</td>
<td>3.1, 1.1, 1.1, 3.1</td>
<td>1.3, 1.1, 1.1, 1.3</td>
</tr>
<tr>
<td>-3.3, 1.3, -1.3, -3.3</td>
<td>-3.3, -1.3, -1.3, -3.3</td>
<td>3.3, 1.3, 1.3, 3.3</td>
<td>-3.3, 3.1, 3.1, 3.3</td>
</tr>
</tbody>
</table>

Figure 4.9: The eight isometric transformed domain blocks of a domain block (D1).

To perform the comparison processes is \( \log_2 8 \).

For executing the eight isometric transformations and selecting the best transformation among them, \((360 \text{ degree rotation}) + (\text{four rotated domains}) + (\text{four reflected domains}) + (\text{comparison}) = 12C + C/2 + 3\) steps are needed. This module is capable of performing the fast rotations using only one domain block without the external memory for all the domain blocks.
Table 4.2: The parameters of the eight isometric transformations $M_f$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>Parameters</th>
<th>$\alpha$, $\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identity</td>
<td>$\alpha = 0$, $\beta = 1$</td>
</tr>
<tr>
<td>2</td>
<td>Rotation through $+90^\circ$</td>
<td>$\alpha = 1$, $\beta = 1$</td>
</tr>
<tr>
<td>3</td>
<td>Rotation through $+180^\circ$</td>
<td>$\alpha = 2$, $\beta = 1$</td>
</tr>
<tr>
<td>4</td>
<td>Rotation through $+270^\circ$</td>
<td>$\alpha = 3$, $\beta = 1$</td>
</tr>
<tr>
<td>5</td>
<td>Reflection about mid-vertical axis</td>
<td>$\alpha = 0$, $\beta = -1$</td>
</tr>
<tr>
<td>6</td>
<td>Reflection and Rotation$-90^\circ$</td>
<td>$\alpha = -1$, $\beta = -1$</td>
</tr>
<tr>
<td>7</td>
<td>Reflection and Rotation$-180^\circ$</td>
<td>$\alpha = -2$, $\beta = -1$</td>
</tr>
<tr>
<td>8</td>
<td>Reflection and Rotation$-270^\circ$</td>
<td>$\alpha = -3$, $\beta = -1$</td>
</tr>
</tbody>
</table>

4.2.4 Validity of Fast Comparison Operation

Let $P_{k,l}(t)$ represent the position of a pixel $(k, l)$ of D1 by counterclockwise step-by-step rotation along the solid-line arrows at time $t$, where $-C+1 \leq k, l \leq C-1$. Figure 4.9 shows the eight isometric transformed domain blocks of a domain block (D1), where we consider that the center of the block is $k = l = 0$ and indices are odd numbers to simplify the description.

The index mapping function $M_f$ of isometric transformation is defined by

$$M_f = \begin{pmatrix} \cos \frac{\pi}{2} \alpha & -\sin \frac{\pi}{2} \alpha \\ \sin \frac{\pi}{2} \alpha & \cos \frac{\pi}{2} \alpha \end{pmatrix} \begin{pmatrix} \beta & 0 \\ 0 & 1 \end{pmatrix}$$

where $\alpha$ and $\beta$ are determined by Table 4.2.

Since $M_f$ maps a pixel $(k, l)$ of the isometric transformed domain to the corresponding position of a range pixel, $T_f(d)(k, l) = d(M_f^{-1}(k, l))$.

Thus, we obtain
4.2. THE PROPOSED FPFIC ARCHITECTURE

\[
\sum_{k,l} |T_f(d)(k,l) - r(k,l)| = \sum_{k,l} |d(k,l) - r(M_f(k,l))|. \tag{4.3}
\]

A domain pixel \((k, l)\) transfers to PP \(P_{k,l}(t)\) which corresponds to the position of a range pixel, so that PP computes \(|r(P_{k,l}(t)) - d(k, l)|\). If \(P_{k,l}(t) = M_f(k, l)\), \(|r(M_f(k, l)) - d(k, l)|\) is computed at PP \(M_f(k, l)\) at time \(t\).

By the connection of counterclockwise rotation, four isometric transformations \(M_f\) are clearly realized at the time \(\frac{C}{2}(f - 1)\). Namely, we have

\[
P_{k,l}(\frac{C}{2}d) = M_{d+1} \begin{pmatrix} k \\ l \end{pmatrix} = \begin{pmatrix} k_d \\ l_d \end{pmatrix}, \quad d = 0, 1, 2, 3. \tag{4.4}
\]

The isometric transformations \(M_f (f = 5, 6, 7, 8)\) with reflection are not realized at a fixed time, but some particular parts of D5-8 which are results of \(M_f(D1)\) are coincided with the shifting movement at some points in time.

We will find the time \(t\) and the pixel \(M_f(k, l) (f = 5, 6, 7, 8)\) of D5-8 such that \(P_{k,l}(t) = M_f(k, l)\).

For \(\frac{C}{2}d \leq t < \frac{C}{2}(d + 1)\), \(t\) can be represented as \(t = \frac{C}{2}d + \Delta\), where \(\Delta = 0, 1, \ldots, \frac{C}{2} - 1\). \(P_{k,l}(\frac{C}{2}d + \Delta)\) can be calculated by setting \((k_d, l_d)\) to the base position and moving the positions in the directions as follows:

- IF \((k_d > 0, l_d > 0)\) THEN \((-1, 1)\) : the first quadrant
- IF \((k_d < 0, l_d > 0)\) THEN \((-1, -1)\) : the second quadrant
- IF \((k_d < 0, l_d < 0)\) THEN \((1, -1)\) : the third quadrant
- IF \((k_d > 0, l_d < 0)\) THEN \((1, 1)\) : the fourth quadrant
4.2. THE PROPOSED FPIC ARCHITECTURE

These are coincided with the solid-line connection of data transfers as shown in Figures 4.5 and 4.6. These directions can be represented by \((-\text{sign}(l_d), \text{sign}(k_d))\), where

\[
\text{sign}(x) = \begin{cases} 
1 & \text{if } x \geq 0 \\
-1 & \text{if } x < 0.
\end{cases}
\]  \hspace{1cm} (4.5)

And the moved position is represented as follows:

\[
\begin{pmatrix}
K_d(\Delta) \\
L_d(\Delta)
\end{pmatrix} = \begin{pmatrix}
k_d \\
l_d
\end{pmatrix} + 2\Delta \begin{pmatrix}
-\text{sign}(l_d) \\
\text{sign}(k_d)
\end{pmatrix}.
\]  \hspace{1cm} (4.6)

Since the connection is restricted, the position \((K_d(\Delta), L_d(\Delta))\) may be outside of the considered quadrant. For that case, the position \(P_{k,l}(\frac{C}{2}d + \Delta)\) is calculated with \((k_{d+1}, l_{d+1})\) as the base position, since \(P_{k,l}(\frac{C}{2}d + \Delta) = P_{k,l}(\frac{C}{2}(d+1) - (\frac{C}{2} - \Delta))\).

By the same discussion for Equation (4.6), we have

\[
\begin{pmatrix}
K_{d+1}(\Delta) \\
L_{d+1}(\Delta)
\end{pmatrix} = \begin{pmatrix}
k_{d+1} \\
l_{d+1}
\end{pmatrix} - (C - 2\Delta) \begin{pmatrix}
-\text{sign}(l_{d+1}) \\
\text{sign}(k_{d+1})
\end{pmatrix}.
\]  \hspace{1cm} (4.7)

Since the position \(P_{k,l}(\frac{C}{2}d + \Delta)\) is in the same quadrant of \((k_d, l_d)\) or \((k_{d+1}, l_{d+1})\), the position of pixel at the time \(t = \frac{C}{2}d + \Delta\) \((d = 0, 1, 2, 3, \Delta = 0, 1, \ldots, \frac{C}{2} - 1)\) can be written as

\[
P_{k,l}(\frac{C}{2}d + \Delta) = \begin{cases} 
\begin{pmatrix}
K_d(\Delta) \\
L_d(\Delta)
\end{pmatrix} & \text{if } K_d(\Delta)k_d > 0, \\
\begin{pmatrix}
L_d(\Delta) \\
K_d(\Delta)
\end{pmatrix} & \text{if } L_d(\Delta)l_d > 0,
\end{cases}
\begin{pmatrix}
K'_{d+1}(\Delta) \\
L'_{d+1}(\Delta)
\end{pmatrix} = \begin{cases} 
\begin{pmatrix}
K_{d+1}(\Delta) \\
L_{d+1}(\Delta)
\end{pmatrix} & \text{if } K'_{d+1}(\Delta)k_{d+1} > 0, \\
\begin{pmatrix}
L_{d+1}(\Delta) \\
K'_{d+1}(\Delta)
\end{pmatrix} & \text{if } L'_{d+1}(\Delta)l_{d+1} > 0
\end{cases}
\]  \hspace{1cm} (4.8)
Now, we show that the calculation for the reflected domains can be performed by the rotation of the original domain. Since all the reflected domains are obtained by the rotation of D5, we consider a quadrant \((k, l > 0)\) of D1 and the reflected domain D5. The pixel of D5 is \((\hat{k}, \hat{l})^T = M_5(k, l)^T = (-k, l)^T\) by Equation (4.2), where \(T\) is the transpose of a vector. In order to know the time at which the pixel \((k, l)\) of D1 rotates to the position of the pixel \((\hat{k}, \hat{l})\) of D5, we have an equation against \(t\): \(P_{k,l}(t) = (-k, l)^T\).

We calculate the pixel of D1 rotated to the position of D5 for the case of \(d = 0, 1, 2, 3\).

**Case I** \((0 \leq t \leq \frac{C}{2} - 1, d = 0)\):

From Equation (4.8),

\[
\begin{pmatrix}
-k \\
l
\end{pmatrix}
= \begin{pmatrix}
K_0(\Delta) \\
L_0(\Delta)
\end{pmatrix}
= \begin{pmatrix}
k_0 \\
l_0
\end{pmatrix}
+ 2\Delta \begin{pmatrix}
\text{sign}(l_0) \\
\text{sign}(k_0)
\end{pmatrix} + \begin{pmatrix}
-2\Delta \\
2\Delta
\end{pmatrix}.
\]

Equation (4.9) has no feasible solutions for odd \(k, l > 0\) and consider the second case of Equation (4.8).

\[
\begin{pmatrix}
-k \\
l
\end{pmatrix}
= \begin{pmatrix}
K_1(\Delta) \\
L_1(\Delta)
\end{pmatrix}
= \begin{pmatrix}
k_1 \\
l_1
\end{pmatrix} - (C - 2\Delta) \begin{pmatrix}
\text{sign}(l_1) \\
\text{sign}(k_1)
\end{pmatrix} + \begin{pmatrix}
-2\Delta \\
2\Delta
\end{pmatrix},
\]

therefore,

\[
l = k + C - 2\Delta.
\]

Hence,

\[
\Delta = \frac{1}{2}(k - l + C), \quad t = \frac{C}{2} \cdot 0 + \Delta = \frac{1}{2}(k - l + C).
\]
The conditions for the second case of Equation (4.8):

\[
\begin{align*}
\begin{cases}
K'_1(\Delta)k_1 &= (-l + C - 2\Delta)(-l) > 0 \\
L'_1(\Delta)l_1 &= (k + C - 2\Delta)(k) > 0
\end{cases}
\end{align*}
\tag{4.13}
\]

give a range of \((k, l)\) such that

\[
\begin{cases}
l > C - 2\Delta \\
k > 0
\end{cases}
\tag{4.14}
\]

**Case II** \((C/2 \leq t \leq C - 1, d = 1):\)

From the first case of Equation (4.8),

\[
\begin{align*}
\begin{pmatrix}
-k \\
l
\end{pmatrix} &= \begin{pmatrix}
k_1 \\
l_1
\end{pmatrix} + 2 \begin{pmatrix}
-\text{sign}(l_1) \\
\text{sign}(k_1)
\end{pmatrix} \Delta \\
&= \begin{pmatrix}
-l \\
k
\end{pmatrix} + \begin{pmatrix}
-2\Delta \\
-2\Delta
\end{pmatrix}. 
\end{align*}
\tag{4.15}
\]

Thus,

\[
l = k - 2\Delta.
\tag{4.16}
\]

Hence,

\[
\Delta = \frac{1}{2}(k - l), \quad t = \frac{C}{2} \cdot 1 + \Delta = \frac{1}{2}(k - l + C).
\tag{4.17}
\]

The conditions for the first case of Equation (4.8):

\[
\begin{align*}
\begin{cases}
(-l - 2\Delta)(-l) &> 0 \\
(k - 2\Delta)(k) &> 0
\end{cases}
\end{align*}
\tag{4.18}
\]

give a range of \((k, l)\) such that

\[
\begin{cases}
l > 0 \\
k > 2\Delta
\end{cases}
\tag{4.19}
\]

The second case of Equation (4.8) has no feasible solutions.
4.2. THE PROPOSED FPFIC ARCHITECTURE

Figure 4.10: The position and the time of the pixel of D1 that can be rotated to D5.

Case III \((C \leq t \leq \frac{3C}{2} - 1, d = 2)\):

Case IV \((\frac{3C}{2} \leq t \leq 2C - 1, d = 3)\):

Case III and IV have no feasible solutions for \(k, l > 0\).

Figure 4.10 shows the solutions of Equations (4.11), (4.14), (4.16), and (4.19). We see from Figure 4.10 that each position at its respective time of the pixels of D1 can be rotated to D5. Each pixel \((k, l)\) on these lines can be rotated to \(M_5(k, l) = (\hat{k}, \hat{l})\) of D5 at \(t = \frac{C}{2}d + \Delta\) where \(C = 8\). Therefore, it is clear that all of the pixels in the first quadrant can be used for calculation of D5.

The time at which the computation with the pixels \((k, l)\) and \((\hat{k}, \hat{l})\) is computed
can be written from Equations (4.12) and (4.17) as follows:

\[ t = \frac{1}{2}(k - l + C) \quad \text{if } k, l > 0 . \]  

\[(4.20)\]

**4.3 For the P-pixels-interval Domain Pool**

If higher quality images are needed, you have to use the larger domain pool such as the P-pixels-interval domain pool \( \mathcal{D}_p \). Since the data in domain memory is obtained by sub-sampling, a pixel of the extended domain memory corresponds to the data of 2\( \times \)2 pixels in the range block. So that the overlapping interval parameter \( P \) must be \( P \geq 2 \) and \( P \) is an even number. The number of domain blocks largely depends on \( P \) as follows:

\[ (N_{D_p})^2 = \left( \frac{N - D}{P} + 1 \right)^2, \]

where \( D = 2R \) and \( N = N_R \times R \).
4.4. EXPERIMENTAL RESULTS

The proposed architecture for $\mathcal{D}_P$ can be designed by extending the domain block memory as shown in Figure 4.11. Each PE has the extended right domain memory (ERDM), the extended bottom domain memory (EBDM) and the extended diagonal domain memory (EDDM) to store the extended domain data from other PEs as follows:

$$
    d_{ERDM(i,j)} = E_d(r_{(i+2,j)}, r_{(i+2,j+1)})
$$
$$
    d_{EBDM(i,j)} = E_d(r_{(i,j+2)}, r_{(i+1,j+2)})
$$
$$
    d_{EDDM(i,j)} = E_d(r_{(i+2,j+2)})
$$

(4.22)

where $E_d$ is sub-sampling operator.

These groups of data are transmitted immediately from the three neighboring PEs after the first stage. At the end of the comparison using the half-overlapping domain pool, ERDM, EBDM and EDDM send a column and/or a row line of data to its domain memory. The encoding procedure is repeated until all range blocks have compared themselves against every domain block. The number of steps needed for the comparisons using $\mathcal{D}_P$ is $(\frac{R}{2})^2 \times (N_D)^2$, which is larger than $(N_{D_p})^2$ since the right-most column and bottom PPs include incomplete domain blocks by moving of domain data.

4.4 Experimental Results

This FPFIC architecture is very modular and hierarchical due that FPFIC consists of PE array and each PE has many PPs. Since the FCM of the PE is the most important module, it has been considered in this experiment. FCM consists of $C \times C$ PPs, the eight buffers and CPs as shown in Figure 4.8.
4.4. EXPERIMENTAL RESULTS

We implement a FCM in Verilog HDL to evaluate the features of the architecture. Since MAD is suitable for VLSI implementation with the aim of cutting down on computation time and/or area, MAD is used as the distortion criterion in this experiment.

Figure 4.12 shows the simulation result by Verilog HDL, where RESULT and RESULT-N represent the distortion value of the best domain and the number-id of domain block, respectively. This verification of the description correctness was performed by Verilog-XL of Cadence company. Figure 4.13 shows the Verilog HDL synthesis of FCM by automatic synthesis process of Design Analyzer of Synopsys company, based on ROHM 0.35 \( \mu \text{m} \) CMOS technology. The estimated total number of gates required for FCM is around 83563. Figures 4.12 and 4.13 show the functional validity of the proposed architecture.

The proposed FCM has been laid out using a 0.35 \( \mu \text{m} \) triple metal CMOS process as shown in Figure 4.14. The FCM chip contains 334252 transistors in a die size of 2.9 \( \times \) 2.9 mm\(^2\). Simulation shows that it has the capability to run up to 40 MHz, or 25 ns per cycle.

If every operation is performed in one clock cycle, the number of clock cycles required for encoding one image will be: \((12C + C/2 + 3) \times (N_p)^2\), where \((N_p)^2\) FCM chips are contained in the FP\(\text{FIC} \) system. Table 4.3 gives encoding time estimations based on cycles per operation. The estimation result shows that the FP\(\text{FIC} \) system encodes 256 \( \times \) 256 and 512 \( \times \) 512 images in 5.4 and 21.7 ms, respectively, where the size of range block is 4 \( \times \) 4 image and 40 MHz clock signal. If the frame rate is 30 frames per second, encoding one frame should be
processed in 33.3 ms. Therefore, the FPFIC system with \((N_p)^2\) FCM chips can meet the real time video applications.

The other parts of PE consist of memory module, contractive mapping module, Mux and Dec, which are required small size circuits. If one PE can be implemented in a chip, FIC system must be implemented with many chips in order to encode the image. The number of chips needed for FIC system is the total number of range blocks, \(N_R \times N_R\).
4.5 Summary

We have proposed an efficient VLSI architecture for fixed-size partitioning fractal image compression (FPFIC). The main feature of this architecture is that it is capable of performing the fractal image encoding without the external memory for the fixed domain blocks.

<table>
<thead>
<tr>
<th>Image dimension</th>
<th>pixel</th>
<th>cycles</th>
<th>encoding time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>256×256</td>
<td></td>
<td>217088</td>
<td>0.0054</td>
</tr>
<tr>
<td>512×512</td>
<td></td>
<td>868352</td>
<td>0.0217</td>
</tr>
</tbody>
</table>
4.5. SUMMARY

We have proposed the fast comparison module (FCM) in each processor which can calculate the eight isometric transformations in one full rotation around the center and have shown the validity of the algorithm of FCM. It comes out that by using FCM gains a speed-up of 2 times over the general pipeline processing.

The FPFIC architecture for the P-pixels-interval domain pool has been proposed. Each PE of this architecture has the extended right domain memory (ERDM), the extended bottom domain memory (EBDM) and the extended diagonal domain memory (EDDM) to store the extended domain data from other PEs.

We have implemented FCM with ROHM 0.35 µm CMOS technology. If one PE can be implemented in a chip, FPFIC system must be implemented with \( N_R \times N_R \) chips. The further work is to reduce the number of chips in order to implement FIC.
Figure 4.14: The chip layout of the proposed FCM with 0.35 \( \mu \)m CMOS technology.
Chapter 5

An Architecture for Quadtree-based Partitioning Fractal Image Compression

5.1 Introduction

We have proposed a parallel processing architecture for FPFIC in Chapter 4. The partitioning scheme of FPFIC which is fixed-size partitioning which does not give better performance than flexible-size partitionings. The quadtree partitioning is the most commonly used one based on the flexible-size range blocks. However, it was difficult to realize a VLSI architecture based on flexible-size partitioning.

In this chapter, we propose a parallel processing architecture for the Quadtree-based Partitioning Fractal Image Compression (QPFIC) that uses flexible-size partitioning. This architecture is designed by modifying FPFIC model that uses only local data communication and FCM module. The key idea of this architecture is that the quadtree partitioning encoding can be performed by using the results of fixed-size partitioning encoding. Since large domain blocks consist of small domain blocks, the calculations of distortion for large domain blocks
are performed by using only the maximum depth domain pool extracted from
the smallest range blocks of the neighboring processors. The proposed architec-
ture performs the MAD (mean absolute difference) calculation for the maximum
depth domain pool and computes the MADs for other domain pools by adding
the MADs for the maximum depth domain pool. This architecture has the fast
comparison module of FPFIC architecture that is capable of comparing range
blocks with the eight isometric transformed domain blocks by one full rotation
around the center. In Section 5.3, we show the results of computer simulation to
support the validity of the proposed architecture.

5.2 The Proposed QPFIC architecture

The QPFIC algorithm has several levels of range and domain pools, \( R^d \), \( D^d \),
respectively as shown in Figure 3.1, where \( d \) is a level or depth such that \( d_{\text{min}} \leq d \leq d_{\text{max}} \). To reduce the computation for several domain pools, our algorithm
is considering that the MAD for the maximum depth domain pool \( D^{d_{\text{max}}} \) is a
component of the MAD for other parent domain pools. This architecture is
capable of calculating the MADs for \( D^{d_{\text{max}}} \) using the fast comparison module
proposed in this paper. The results obtained in PEs are propagated up to the
memory for \( D^{d_{\text{max}}-1} \), and then to the memory for other domain pools of other
depth.

The basic structure of QPFIC is a systolic array that is similar to FPFIC
architecture except for the memory module. The memory module of PE consists
of three parts, storing a range-domain blocks data, the best MAD value for \( D^{d_{\text{max}}} \),
Figure 5.1: The block diagram of PE.

and the MADs for the other domain pools. Figure 5.1 shows a block diagram of PE that has a fast comparison module and a memory module.

The encoding procedure of the proposed architecture consists of three phases listed below:

- Phase 1: Generation of $\mathcal{D}^{d_{\text{max}}}$ and the MAD calculation for $\mathcal{D}^{d_{\text{max}}}$, and the MAD calculation for $\mathcal{D}^{d_{\text{max}-1}}$ using the results for $\mathcal{D}^{d_{\text{max}}}$

- Phase 2: The MAD calculation for other domain pools

- Phase 3: Quadtree-based encoding by the results of the MAD calculation

5.2.1 Generation of $\mathcal{D}^{d_{\text{max}}}$ and the MAD calculation for $\mathcal{D}^{d_{\text{max}}}$ (First Part of Phase 1)

Phase 1 is the same process of FPFIC architecture described in Section 4. Assume that there are $N_p \times N_p$ PEs and the total number of PEs is equal to the number of the $d_{\text{max}}$ depth range blocks, $N_p = N_R$. The total number of the half-overlapping
domain blocks is \((N_p - 1)^2\) because the overlapping interval is \(D/2 = R\). We assume that each range block is loaded into each PE. A domain block is drawn from four neighboring range blocks. To save memory, the domain blocks are contracted in advance and stored in PEs. Each range block is compared with its corresponding domain block in parallel and all domain blocks are shifted to the next PEs by ring connection as shown in Figure 4.4. This architecture is able to perform the fast isometric transformations by using FCM of FPIFIC architecture.

5.2.2 The MAD calculation for \(\mathcal{D}^{d_{\text{max}}^{-1}}\) (Second Part of Phase 1)

Note that a MAD for the maximum-depth domain pool \(\mathcal{D}^{d_{\text{max}}}\) is a component of a MAD for other parent domain pools. The MAD calculation for a large domain block is represented by the summation of the MAD calculations for its smaller domain blocks due to the property of the MAD calculation. For example, Equation (2.11) is represented such that

\[
\delta(\tilde{r}_{i,j}, r_{i,j}) = \sum_{k=1}^{\frac{R}{2}} \sum_{l=1}^{\frac{R}{2}} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| + \sum_{k=1}^{\frac{R}{2}} \sum_{l=\frac{R}{2}+1}^{R} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| \\
+ \sum_{k=\frac{R}{2}+1}^{R} \sum_{l=1}^{\frac{R}{2}} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| + \sum_{k=\frac{R}{2}+1}^{R} \sum_{l=\frac{R}{2}+1}^{R} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| .
\]

Since each contracted domain block of \(\mathcal{D}^{d_{\text{max}}^{-1}}\) is composed of the four contracted domain blocks of \(\mathcal{D}^{d_{\text{max}}}\), the results of the MADs for \(\mathcal{D}^{d_{\text{max}}^{-1}}\) are obtained by adding four results of the MADs for \(\mathcal{D}^{d_{\text{max}}}\). It needs all the MADs for \(\mathcal{D}^{d_{\text{max}}}\) for the MAD calculation for \(\mathcal{D}^{d_{\text{max}}^{-1}}\). Therefore, while the MAD for \(\mathcal{D}^{d_{\text{max}}}\) is
Figure 5.2: The $d - 1$ depth range and domain block consist of four $d$ depth range and domain blocks. (a) range blocks. (b) domain blocks.

calculated on Phase 1, it is transmitted to the memory to calculate the MAD for $D_{d_{max}}^{-1}$. We have to know the MAD for what domain block of $D_{d_{max}}^{-1}$ is calculated at a particular time of Phase 1.

Note that the $d - 1$ depth range and domain block consist of four $d$ depth range and domain blocks, respectively. The range and domain blocks are represented as the following equations, where the domain pool is the half-overlapping domain pool and the contracted domain blocks (see Figure 5.2).
where $I = \lfloor \frac{d}{2} \rfloor$ and $J = \lfloor \frac{d}{2} \rfloor$ are indices of the $d-1$ depth blocks and $i$, $j$ are indices of the $d$ depth blocks.

The isometric transformations of $d_{I,J}^{d-1}$ are represented:

\[
T_1 (d_{I,J}^{d-1}) = \begin{bmatrix} T_1 (d_{i,j+2}^d) & T_1 (d_{i+2,j}^d) \\ T_1 (d_{i,j+2}^d) & T_1 (d_{i+2,j}^d) \end{bmatrix},
T_2 (d_{I,J}^{d-1}) = \begin{bmatrix} T_2 (d_{i+2,j+2}^d) & T_2 (d_{i,j+2}^d) \\ T_2 (d_{i+2,j+2}^d) & T_2 (d_{i,j+2}^d) \end{bmatrix},
\]

\[
T_3 (d_{I,J}^{d-1}) = \begin{bmatrix} T_3 (d_{i+2,j+2}^d) & T_3 (d_{i,j+2}^d) \\ T_3 (d_{i+2,j+2}^d) & T_3 (d_{i,j+2}^d) \end{bmatrix},
T_4 (d_{I,J}^{d-1}) = \begin{bmatrix} T_4 (d_{i,j+2}^d) & T_4 (d_{i,j}^d) \\ T_4 (d_{i,j+2}^d) & T_4 (d_{i,j}^d) \end{bmatrix},
\]

\[
T_5 (d_{I,J}^{d-1}) = \begin{bmatrix} T_5 (d_{i+2,j+2}^d) & T_5 (d_{i,j+2}^d) \\ T_5 (d_{i+2,j+2}^d) & T_5 (d_{i,j+2}^d) \end{bmatrix},
T_6 (d_{I,J}^{d-1}) = \begin{bmatrix} T_6 (d_{i,j+2}^d) & T_6 (d_{i,j}^d) \\ T_6 (d_{i,j+2}^d) & T_6 (d_{i,j}^d) \end{bmatrix},
\]

\[
T_7 (d_{I,J}^{d-1}) = \begin{bmatrix} T_7 (d_{i+2,j+2}^d) & T_7 (d_{i,j+2}^d) \\ T_7 (d_{i+2,j+2}^d) & T_7 (d_{i,j+2}^d) \end{bmatrix},
T_8 (d_{I,J}^{d-1}) = \begin{bmatrix} T_8 (d_{i,j+2}^d) & T_8 (d_{i,j}^d) \\ T_8 (d_{i,j+2}^d) & T_8 (d_{i,j}^d) \end{bmatrix},
\]

for $0 \leq I, J \leq \frac{N_p}{2} - 2$ and $0 \leq i, j \leq N_p - 4$, (5.3)
\[ \delta(r_{i,j}^{d-1}, T_2(d_{i,j}^{d-1})) = \delta(r_{i,j}^{d}, T_2(d_{i+1,j}^{d})) + \delta(r_{i+1,j}^{d}, T_2(d_{i+2,j+2}^{d})) + \delta(r_{i+1,j+1}^{d}, T_2(d_{i,j+2}^{d})), \]

(5.4)

where the MAD calculation is used as the distortion criterion.

All domain blocks are shifted by ring connection. Each PE computes the distortion between its range block and the domain block shifted from its neighboring PE.

The PE position \((p, q)\) at which the domain block \(d_{i,j}^{d}\) is computed at \(t\) time is represented by Equation (5.5):

\[
\begin{align*}
  p &= (t + P_{do}) \mod N_p \\
  q &= \left\lfloor (t + P_{do})/N_p \right\rfloor \mod N_p,
\end{align*}
\]

(5.5)

where \(P_{do} = i + j \times N_p\).

Equation (5.6) shows the time at which the domain block \(d_{i,j}^{d}\) is calculated at PE \((p, q)\).

\[
\begin{align*}
  t &= P_{pe} - P_{do} \quad \text{If } P_{pe} \geq P_{do} \\
  t &= P_{pe} - P_{do} + N_p \times N_p \quad \text{If } P_{pe} < P_{do},
\end{align*}
\]

(5.6)

where \(P_{pe} = p + q \times N_p\) and \(P_{do} = i + j \times N_p\).

Note that each PE is capable of computing the MADs for \(D_{d_{max}-1}\) with \(D_{d_{max}}\) by Equations (5.2)-(5.6). Each PE is capable of obtaining the information for
the current domain block \( d_{i,j}^d \) in each PE by Equations (5.5) and (5.6). If \( i \) and \( j \) are even numbers, this domain block is a component of the \( d - 1 \) depth domain blocks, \( d_{I,J}^{d-1} \), \( d_{I-1, J}^{d-1} \), \( d_{I, J-1}^{d-1} \), and \( d_{I-1, J-1}^{d-1} \) where \( I = \lfloor \frac{i}{2} \rfloor \) and \( J = \lfloor \frac{j}{2} \rfloor \). Since the isometric transformations of them are presented by Equation (5.3), each PE is capable of adding the distortion result of \( d_{i,j}^d \) to the result for the \( d - 1 \) depth isometric transformed domain blocks.

The total steps needed for Phase 1 are (the number of PEs)\( \times \) (the fast distortion calculation for for one domain block) = \( (N_p)^2 \times (12C + \frac{C}{2} + 3) \).

5.2.3 The MAD calculation for other domain pools (Phase 2)

Each PE has the memory to store the information of the best matched domain block of \( D_{\text{max}} \) and the MADs for all the domain blocks \( D_{\text{max}^{-1}}, \ldots, D_{\text{min}} \). Since one block of the upper level domain pool contains four blocks of the lower level domain pool, the MADs for the upper level domain pool are obtained by the EA (Error Adder) modules that add four MADs for the lower level domain pool (see Figure 5.3). First, each EA module adds the MADs for \( D_{\text{max}^{-1}} \) from four neighboring PEs (denoted by circles) and finds the best domain for \( D_{\text{max}^{-1}} \). The upper EA modules add the MADs from the lower EA modules and find the best domain for them. To obtain the MADs for all the domain pools except \( D_{\text{max}} \), \( d_{\text{max}} - d_{\text{min}} \) steps are needed.
5.2. THE PROPOSED QPFIC ARCHITECTURE

Figure 5.3: The MAD calculation for other domain pools by the EA modules.

5.2.4 Quadtree-based encoding by the results of the MAD calculation (Phase 3)

The results of the MAD calculation for all the domain pools are stored in PEs and EAs. The results of the MADs in each EA module are compared and the minimum MAD is selected. If the minimum MAD value is above a preselected threshold, then the block, corresponding to the value, is divided into four blocks, corresponding to the connected four EA modules, and the process is repeated. If the minimum MAD value is below the threshold, the information of the optimal domain block is stored. When the depth of the quadtree is $d_{max}$, the optimal domain block of $D^{d_{max}}$ is the pre-calculated domain in PE. $d_{max} - d_{min}$ steps are
needed to perform Phase 3 since the calculation of Phase 3 can be performed in parallel for each range block.

The $s, o$ values of the optimal domain block are computed by using $\sum r d, \sum d^2$ and $\sum r$ obtained from $d_{max}$ level (see Equations 2.16 and 2.17). Equations 5.7 and 5.8 show that the $s, o$ value of the block can be calculated by adding four $\sum r d, \sum d^2$ and $\sum r$ values corresponding to the connected four blocks.

$$s_{i,j}^{d-1} = \frac{\sum r_{i,j}^d i_{i,j}^d + \sum r_{i+1,j}^d i_{i+1,j}^d + \sum r_{i,j+1}^d i_{i,j+1}^d + \sum r_{i+1,j+1}^d i_{i+1,j+1}^d}{\sum(d_{i,j}^d)^2 + \sum(d_{i+1,j}^d)^2 + \sum(d_{i,j+1}^d)^2 + \sum(d_{i+1,j+1}^d)^2}, \quad (5.7)$$

$$a_{i,j}^{d-1} = \frac{1}{R^2} \left( \sum r_{i,j}^d + \sum r_{i+1,j}^d + \sum r_{i,j+1}^d + \sum r_{i+1,j+1}^d \right), \quad (5.8)$$

where $(I, J)$ and $(i, j)$ are indices of the $d - 1$ and $d$ depth blocks.

These $s, o$ values are components of code and are only used in decoding process while these are not used in encoding process. The MAD values are the approximated values which are obtained only adding the results for lower level. All of the selected optimal domain blocks may not be optimal because the MAD values used to select the best are the approximated values calculated with the bottom depth blocks. So that, we will show the validity of the encoding process by experiments.

### 5.3 Experimental Results

In this section, we simulate the proposed QPFIC architecture using computer simulation. We implemented the encoding algorithm of this architecture with C...
Figure 5.4: The simulation result of the proposed algorithm for 128 × 128 Lena image (26.42 PSNR, 1.29 bpp).

language. This program has three phases to encode an image corresponding to the encoding process of the proposed architecture. Each PE operates the distortion calculation and the moving of domain block to next PE in sequence.

Figure 5.4 shows the results encoded by the programs of QPFIC architecture for 128 × 128 Lena image.

Figure 5.5 shows PSNR versus the bit rate results for 128 × 128 Lena image using a conventional fixed-size, quadtree partitionings and the proposed algorithm used in QPFIC. The proposed algorithm is below the curve of quadtree partitioning because the proposed algorithm used the approximated MAD values.

The proposed algorithm gives better performance than fixed-size partitioning and its computational complexity is the same as that of the fixed-size partitioning because the MAD calculation is needed only for the maximum depth domain pool.
5.4. Summary

It was difficult to realize a VLSI architecture based on flexible-size partitioning while flexible-size partitionings such as the quadtree partitioning yield better performance than fixed-size partitionings.

We have proposed the parallel processing architecture for QPFIC which uses flexible-size partitioning. The main feature of this architecture is that its structure is similar to the architecture for FPFIC which has FCM module and local data communication.

This architecture is capable of performing the encode scheme based on flexible-size partitioning better than fixed-size partitioning with the computation com-
plexity of FPFIC.

The computer simulation of the encoding algorithm used in this architecture shows the validity of the proposed architecture. The results of the computer simulation show that it gives better performance than the architecture based on FPFIC. The further work is to reduce the memory of PE to store the results of the MAD calculation.
Chapter 6

Concluding Remarks

6.1 Conclusions

The main goal of this thesis is the development of fractal image coding and the design of its parallel processing architecture. The contributions of this thesis can be summarized as follows.

In Chapter 3, we have proposed alternating binary-tree partitioning (ABP) for fractal image coding. ABP is based on binary-tree partitioning and yields better performances than fixed-size or quadtree partitionings. And ABP is an efficient partitioning scheme for the special-purpose hardware for high-speed fractal image coding because its data flow is regular. To reduce the bits needed to represent the positions of domain blocks, we fixed the range shapes and the positions of the partitioning. The experimental results show that ABP yields lower bit rates and higher qualities than quadtree partitioning for all test images. ABP is an efficient partitioning scheme for the special-purpose hardware for FIC because its data flow is regular and ABP will achieve good results for any images.

In Chapter 4, we have proposed an efficient VLSI architecture for fixed-size
partitioning fractal image compression (FPFIC), which uses only local communication. The main feature of this architecture is that it is capable of performing the fractal image encoding without the external memory for the fixed domain blocks. The proposed fast comparison module (FCM) in each processor which can calculate the eight isometric transformations by one full rotation around the center. This module is capable of computing the distortions between the reflected domains and range blocks using the results of the distortion calculation for the rotated domains by the data dependence between domains. The FPFIC architecture for the P-pixels-interval domain pool has been proposed. Each PE of this architecture has the extended right domain memory (ERDM), the extended bottom domain memory (EBDM) and the extended diagonal domain memory (EDDM) to store the extended domain data from other PEs. We have implemented FCM with ROHM 0.35 μm CMOS technology. The FCM chip contains 334252 transistors in a die size of 2.9 × 2.9 mm². Simulation shows that it has the capability to run up to 40 MHz, or 25 ns per cycle. If one PE can be implemented in a chip, FIC system must be implemented with $N_R \times N_R$ chips.

In Chapter 5, we have proposed a parallel processing architecture for the quadtree-based partitioning fractal image compression (QPFIC) which uses flexible-size partitioning. This architecture is designed by modifying FPFIC model in order to use only local data communication and FCM module. The main feature of this architecture is that its structure is similar to the architecture for FPFIC which has FCM module and local data communication. Since large domain blocks consist of small domain blocks, the calculations of distortion for all kinds of do-
main blocks are performed by using only the maximum depth domain pool which is extracted from the smallest range blocks of the neighboring processors. This architecture performs the MAD calculation for the maximum depth domain pool and computes the MADs for other domain pools by adding the MADs for the maximum depth domain pool. This architecture has the FCM modules of FPFIC architecture which is capable of comparing range blocks with the eight isometric transformed domain blocks by one full rotation around the center. The computer simulation for 128 × 128 Lena image shows the proposed parallel processing architecture for QPFIC gives better performance than fixed-size partitioning used in the architecture for FPFIC.

6.2 Future Work

For the architecture for FPFIC presented in Chapter 4, the experimental results show that FPFIC system must be implemented with many chips. The number of chips needed in FPFIC system depends on the number of range blocks. Assume that the original image we want to encode is an 256 × 256 pixel image and $R=4$ and $D=8$. The total number of range blocks is $(64)^2$ and then FPFIC system has 4096 chips. One may try to reduce the area complexity of the FPFIC architecture.

The encoding algorithm of the QPFIC architecture presented in Chapter 5 is not better than quadtree partitioning because the MAD values are approximated from the MAD results for the bottom depth domain blocks. For the future, work should be done on the MAD calculation without slowing down the encoding time.
Appendix A

Original Images for Experiment
Figure A.1: $512 \times 512$ Lena image.
Figure A.2: 512 × 512 Boat image.
Figure A.3: 512 × 512 Peppers image.
Figure A.4: 512 × 512 Girl image.
Figure A.5: 512 × 512 Mandrill image.
Bibliography


1-D and 2-D Discrete Cosine Transforms,” IEEE Trans. Circuits and Systems


[46] “VLSI Architectures for Block Matching Algorithms Using Systolic Arrays,”

tor Quantization,” IEEE Trans. Circuits and Systems for Video Technology,
vol. 6, no. 4, pp. 389-398, Aug. 1996.


Block-Matching Motion Estimation,” IEEE Trans. Circuits and Systems for
List of Publications

Journal Papers


International Conferences

LIST OF PUBLICATIONS


Domestic Conferences


Acknowledgments

I am indebted to a number of individuals who, either through discussions or by providing the facilities for my study, have contributed to the preparation of this thesis.

I would like to express my appreciation to Professor Hirotomo Aso of the Department of Electrical and Communication Engineering in Tohoku University, for his capable supervision in this research and unforgettable kind help over the course of my doctoral research in Tohoku University.

I am also indebted to Professor Kenichi Abe and Professor Masayuki Kawamata, who have served as members of my doctoral dissertation examination committee, for their highly valuable comments and enlightening discussions.

I am specially grateful to Associate Professor Shin’ichiro Omachi of the Department of Electrical and Communication Engineering in Tohoku University, for his continuous comments and care about my study.

I am grateful to Mr. Masanori Fujibayashi of the Department of Electronic Engineering in Tohoku University, for his helping to simulate the FCM module.

In addition, I take this opportunity to express my sincere appreciation to all members of Aso Laboratory in Tohoku University, for their close collaborations and kind assistances.

Finally, I want to express my special thanks to God.
Index

A
A. Jacquin ...................... 21
adaptive partitionings ........ 36
affine transformation .......... 14
alternating binary-tree partitioning (ABP) 15, 40, 88
ASIC ........................... 15
attractor ....................... 23

B
Barnsley ....................... 13
binary-tree partitioning ...... 38
bit rate ........................ 42

C
C language ...................... 84
Cauchy sequence ................ 23
collage theorem ............... 22
contractive image transformation 23
contractivity factor .......... 22

D
Data compression techniques .... 11
data independence ............ 16, 54
decompression ................ 12
Digital image processing ....... 11
discrete cosine transform (DCT) . 12
distortion criterion .......... 81
domain .......................... 21
domain blocks ................. 14, 25
domain pool .................... 25

E
EA (Error Adder) ............... 82
extended bottom domain memory (EBDM) 69, 73, 89
extended diagonal domain memory (EDDM) 69, 73, 89
extended right domain memory (ERDM) 69, 73, 89
INDEX

F
fast comparison module ........... 76
fast comparison module (FCM) .48, 55, 73, 89
fast-search ..................... 25
fixed point ..................... 23
fixed-size partitioning ....... 15, 36, 75
fixed-size partitioning fractal image compression (FPFIC) 16, 48, 72, 89
fixed-spacing domain pool .... 25
flexible-size partitioning .... 17, 75
flexible-size partitionings .... 75
Fractal Engine ................. 16
Fractal Image Coding (FIC) . 22, 26
fractal image coding (FIC) ...... 48
Fractal image compression .... 13
Fractal Image Decoding (FID) . 31
fractals ....................... 20
full-search .................... 25

G
gray scale ..................... 24

H
half-overlapping domain pool 25, 50, 79
Human Visual System .......... 14
HV partitioning .............. 15, 37

I
IFS .......................... 20, 22
index mapping function ....... 62
Internet ........................ 11
inverse problem ............... 23
isometric transformation .... 25
Iterated Function System .... 20
iterated function systems .... 13

J
Jacquin ........................ 13
Joint Photographic Experts Group 12
JPEG standard ............... 12
Julia sets .................... 20

L
Lipschitz .................... 22
Lipschitz factor ............... 22
lossless ....................... 12
lossy ......................... 12
lossy image compression .... 12
INDEX

M
M. Barnsley ...................... 21
MAD .......................... 76, 90
MAD (mean absolute difference) ... 17
Mandelbrot ...................... 20
maximum depth domain pool . 76, 90
maximum-depth domain pool .... 17
mean absolute difference (MAD) ... 27
mean square error (MSE) ........ 27
O
offset .......................... 25
overlapping interval parameter ... 25
P
P-pixels-interval domain pool 68, 73, 89
parallel architecture .......... 14
parallel processing .......... 16
Partitioned Iterated Function System (PIFS) .......... 21
Peak Signal-to-Noise Ratio (PSNR) ........ 28
pipelined architecture .......... 16
pixel processors (PPs) .......... 55
portable video telecommunication 12
processor elements (PEs) ...... 49
PSNR .......................... 13, 29
Q
quadtree partitioning .... 15, 17, 36
Quadtree-based Partitioning Fractal Image Compression (QPFIC) 75
quadtree-based partitioning fractal image compression (QPFIC) 17, 89
R
range block .................... 22
range blocks .................. 14, 24
range pool .................... 25
resolution independence .... 32
ring connection ............ 53, 78
S
scaling factor .................. 25
self-similarity .............. 22
Sierpinski Triangle .......... 20
special-purpose hardware . 35, 44, 88
systolic array .............. 16, 49, 76
INDEX

T
TV transmission .................12

V
vector quantization .............13
Verilog HDL .....................69
Verilog-XL .......................70
video conferencing .............12
virtual codebook .................13
VLSI architecture..............25, 45, 75
VLSI design ......................15
VLSI implementation ..........44, 70
VLSI technology ...............49