A Parallel Architecture for Quadtree-based Fractal Image Coding

Shinhaeng Lee, Shin'ichiro Omachi and Hirotomo Aso
Department of Electrical and Communication Engineering,
Graduate School of Engineering, Tohoku University,
Aoba, Aramaki, Aoba-ku, Sendai, 980-8579, Japan
TEL : (+81)22-217-7088, FAX : (+81)22-263-9418,
E-mail address : {lee,machi,aso}@aso.ecei.tohoku.ac.jp

Abstract

This paper proposes a parallel architecture for quadtree-based fractal image coding. This architecture is capable of performing the fractal image coding based on quadtree partitioning without the external memory for the fixed domain pool. Since a large domain block consists of small domain blocks, the calculations of distortion for all kinds of domain blocks are performed by the summation of the distortions for the maximum-depth domain pool which is extracted from the smallest range blocks of the neighbor processors. Fast comparison module is proposed for this architecture. This module can compute the distortions between range blocks and their eight isometric transformations by one full rotation around the center.

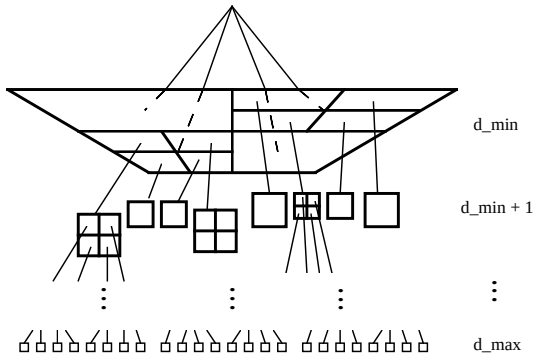

Figure 1. An example of the quadtree partitioning.

1. Introduction

Fractal image compression has recently received great attention and yields high compression ratios and resolution independence in image decompression [1]-[10]. These techniques involve an approach to compression quite different from standard transform coder-based methods. In the fractal image encoding phase, the original image must be divided into range and domain blocks, and the best matching domain block must be found for each range block. The compression process is finished by storing only the descriptions of these transformations. The drawback of fractal image compression is a long encoding time, due to the large amount of comparisons between domain and range blocks. To meet high performance, ASICs are required for high-speed fractal image coding.

A few dedicated architectures proposed have utilized global data communication for providing domain blocks to all the processors [11]-[14]. As the number of processors increases, expanding non-local communication paths is difficult without slowing down the system clock. A parallel architecture for fractal image coding using local communication paths was proposed by the authors [15]. The partitioning scheme of the architecture is fixed-size partitioning which is not capable of yielding better performances than flexible-size partitionings.

In this paper, we propose a parallel architecture for quadtree-based fractal image coding which is a flexible-size partitioning. Since a large domain block consists of small domain blocks, the calculations of distortion for all kinds of domain blocks are performed by using only *the maximum-depth domain pool* which is extracted from the smallest range blocks of the neighbor 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. We also propose the fast comparison module for this architecture. This module is capable of comparing range blocks with the eight isometric transformations of domain blocks by one full rotation around the center. This fast comparison module could

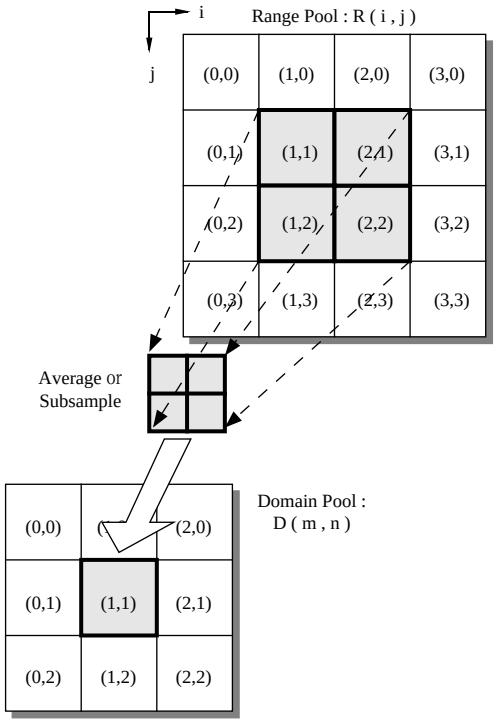

Figure 2. The generation of the half-overlapping domain pool.

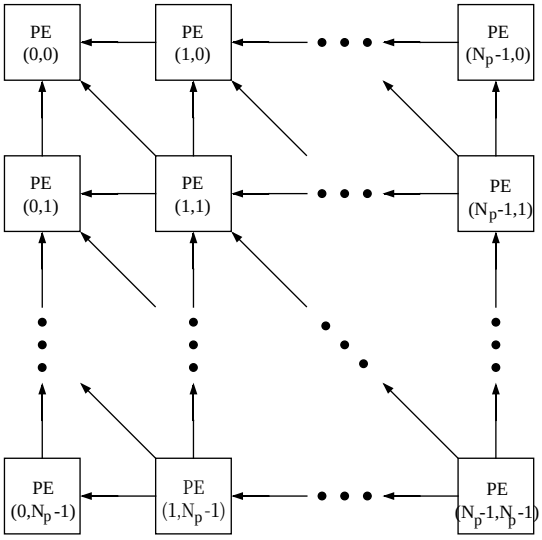

Figure 3. The PE connection for the generation of domain pool.

be used in many image processing applications that contain isometric transformations.

2. The Fractal Image Coding Algorithm

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

$$\mu_{orig} = \bigcup_{i,j=1}^{N_R} r_{i,j}, \quad r_{i,j} \bigcap r_{\hat{i},\hat{j}} = \emptyset \quad \text{for } (i,j) \neq (\hat{i},\hat{j}), \tag{1}$$

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 design an 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 $D/2$ 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. 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 $t_k, 1 \leq k \leq 8$, and spatial contraction S. An isometric transformation t_k maps a square block to one of eight isometries obtained from compositions of reflections and 90 degree rotations.

The result of applying this mapping is an approximation to the $(i,j)^{th}$ range block, $\tilde{r}_{i,j}$ as follows:

$$\tilde{r}_{i,j} = s_{i,j} t_{I(i,j)}(S(d_{A(i,j)})) + o_{i,j}, \tag{2}$$

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 a set of possible isometric transformations.

The encoding process is determining the parameters in equation (2) for all range blocks such that the distortion between each range block and its approximation, $\delta(\tilde{r}_{i,j}, r_{i,j})$, is minimized. Common distortion criteria include the mean square error (MSE) and the mean absolute difference (MAD) which is used in this paper as follows:

$$\delta(\tilde{r}_{i,j}, r_{i,j}) = \sum_{k=1}^{R} \sum_{l=1}^{R} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| \tag{3}$$

where $r_{i,j}(k,l)$ is the pixel (k,l) of the range block $r_{i,j}$.

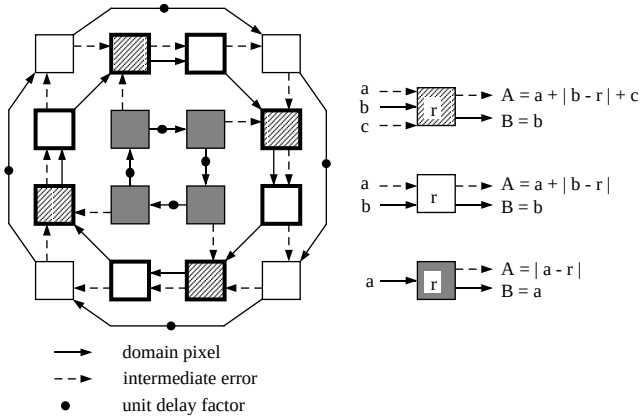

Figure 4. The structure of the fast comparison module in the case of $C = 4$.

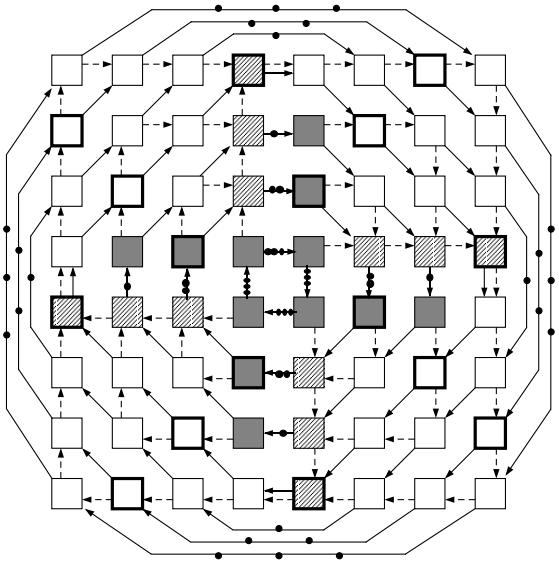

Figure 5. The structure of the fast comparison module in the case of $C = 8$.

In order to simplify the architecture and to increase the performance speed, the MAD distortion criterion is adopted in this paper [11].

In the encoding phase, the most computationally intensive operations are the computation of distortions between range and transformed domain blocks. The total number of computations of distortions is $(N_R)^2 \times (N_D)^2 \times 8$. Assume that the original image we want to encode is an 256×256 pixel image and that $R = 4$, and $D = 8$. The total number of range-domain comparisons is $(64)^2 \times (63)^2 \times 8 = 1.30 \times 10^8$. It is noticed that the dedicated VLSI architecture for fractal image coding is needed.

3. The Quadtree-based Fractal Image Coding

The fixed-size range partitioning is a very simple scheme. 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 [5] 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. The root of the tree is the initial image. If the MAD 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 four sub-blocks, and this process is repeated. If the MAD 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) and a quadtree level d.

4. The Proposed Parallel Architecture

Assume that there are $N_p \times N_p$ PEs and the total number of PEs is equal to the number of 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 $D/2 = R$.

The quadtree-based fractal image coding algorithm has several levels of range and domain pools $\mathcal{R}^d$, $\mathcal{D}^d$ for $d\_min \leq d \leq d\_max$, respectively (see Figure 1). To reduce the computation for several domain pools, our algorithm is based on the fact that a MAD for the maximum-depth domain pool $\mathcal{D}^{d\_max}$ is a component of a MAD for other parent domain pools. This architecture is capable of calculating MADs of $\mathcal{D}^{d\_max}$ using the fast comparison module. The results obtained in PEs are propagated up to the memory for $\mathcal{D}^{d\_max-1}$, and then to the memory for other domain pools of other depth.

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

- Phase 1 : Generation of $\mathcal{D}^{d\_max}$ and the MAD calculation for $\mathcal{D}^{d\_max}$, and the MAD calculation for $\mathcal{D}^{d\_max-1}$ with $\mathcal{D}^{d\_max}$
- Phase 2 : The MAD calculation for other domain pools
- Phase 3 : Quadtree-based encoding by the results of the MAD calculation

4.1. Generation of $\mathcal{D}^{d\_max}$ and the MAD calculation for $\mathcal{D}^{d\_max}$ (Phase 1)

We assume that each range block is loaded into each PE. Figure 2 shows an example of extracting a domain pool in the case of $N_R = 4$. A domain block is drawn from four neighbor range blocks. To save the memory, the domain blocks are contracted in advance and are stored in PEs.

Figure 3 shows the connection of PEs for generation of domain pool $\mathcal{D}^{d\_max}$. The directions of the data flow in each PE are determined by data dependence.

In the literature, the architectures have special memory modules to store the domain pool and memory bandwidth becomes a bottleneck since all PE's access the same memory to receive domain blocks. In this paper, we propose a systolic architecture which resolves this problem by using only local data communication. Each range block is compared with a single domain block in parallel and all domain blocks are shifted to the next PEs by ring connection. The data independency permits the computation of eight comparisons between one range block and the transformed domain blocks in parallel by means of a dedicated hardware architecture [11]. 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 eight isometric transformations without the external memory for domain blocks. To implement fast isometric transformations, this architecture performs fast rotation of a domain block using shifting to the next cell without buffers needed to load and drawn out another block. The proposed architecture consists of $C \times C$ pixel processors (PP's) which calculate the MAD between two pixels.

Figs. 4 and 5 illustrate the structures of the fast comparison modules which are capable of executing eight isometric transformations on domain blocks and selecting the best transformation among them in the case of $C = 4, 8$, respectively. This module performs eight isometric transformations by one full rotation around the center. The unit delay factors are utilized to create a proper movement of data. Each PP in the fast comparison module has one range block pixel and one domain block pixel. 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 MAD between the two pixels, and sends the intermediate result to the next PP along the dotted-line arrow.

In Figure 6a, the calculation of MAD between a range block and a domain block is performed. 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 PPs send 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 four rotated domain blocks. The results of the MAD calculation of 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 distortion calculation (see Figure 6b). After the 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 6c). When the degree of the rotation is 90, the calculation of the absolute difference between a range block and the rotated domain block is performed (see Figure 6d). $2C$ steps are usually needed to exchange one domain block loaded in PP's 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 $4(C/2)$.

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, and are fed into four buffers, respectively as shown in Figure 7. To obtain the results of the four reflected domain blocks, $2C$ steps are needed. The eight results within the buffers are compared to each other using comparators (CPs) to find the smallest value. The number of steps required to perform the comparison processes is $\log_2 8$.

For executing the eight isometric transformations and selecting the best transformation among them, (360 degree rotation) + (four rotated domains) + (four reflected domains) + (comparison) = $2C + 8C + 2C + 3$ steps are needed. This module is capable of performing the fast rotations using only one domain block without the external memory for all domain blocks.

4.2. The MAD calculation for $\mathcal{D}^{d\_max-1}$ with $\mathcal{D}^{d\_max}$ (Phase 1)

Note that a MAD for the maximum-depth domain pool $\mathcal{D}^{d\_max}$ 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 small domain blocks due to the property of the MAD calculation. For example, equation (3) is represented such that

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

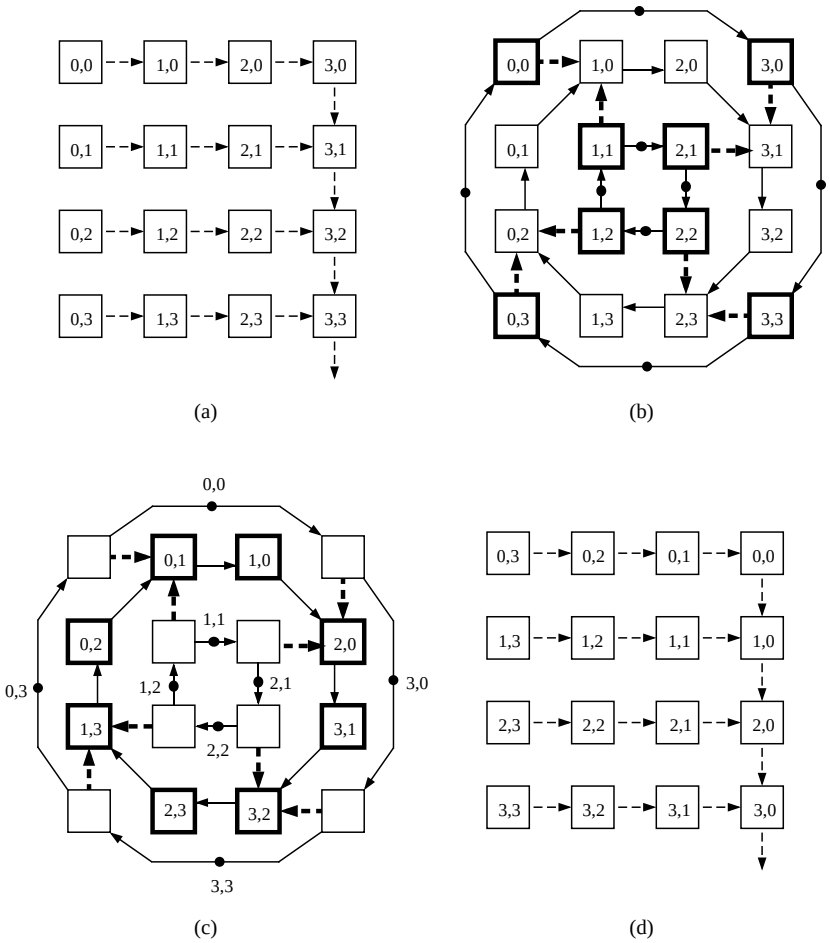

Figure 6. An example of rotation operation in the case of $C = 4$. Indices indicated in PP's are the coordinates of the original pixels. (a) The calculation of the distortion between a range and a domain block. (b) The PPs denoted by the thick-line compute the distortion. (c) The current results are shifted along the dotted-line arrow and added to the intermediate results. (d) The same process is repeated for the rotated domain block.

$$+ \sum_{k=R/2+1}^{R} \sum_{l=1}^{R/2} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)| + \sum_{k=R/2+1}^{R} \sum_{l=R/2+1}^{R} |\tilde{r}_{i,j}(k,l) - r_{i,j}(k,l)|. \quad (4)$$

Since a domain block of $\mathcal{D}^{d\_max-1}$ is composed of the four contracted domain blocks of $\mathcal{D}^{d\_max}$, the results of the MADs of $\mathcal{D}^{d\_max-1}$ are obtained by adding some results of the MADs of $\mathcal{D}^{d\_max}$. It needs all the MADs of $\mathcal{D}^{d\_max}$ for the MAD calculation of $\mathcal{D}^{d\_max-1}$. Therefore, when the MAD of $\mathcal{D}^{d\_max}$ is calculated on Phase 1, it is transmitted to the memory for the MAD of $\mathcal{D}^{d\_max-1}$. We have to know the MAD of what domain block of $\mathcal{D}^{d\_max-1}$ 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 8).

$$r_{I,J}^{d-1} = \left[r_{i,j}^{d}, r_{i+1,j}^{d}, r_{i,j+1}^{d}, r_{i+1,j+1}^{d}\right], \quad \text{for } 0 \le I, J \le \frac{N_p}{2} - 1 \text{ and } 0 \le i, j \le N_p - 2,$$
$$d_{I,J}^{d-1} = \left[d_{i,j}^{d}, d_{i+2,j}^{d}, d_{i,j+2}^{d}, d_{i+2,j+2}^{d}\right], \quad \text{for } 0 \le I, J \le \frac{N_p}{2} - 2 \text{ and } 0 \le i, j \le N_p - 4, \quad (5)$$

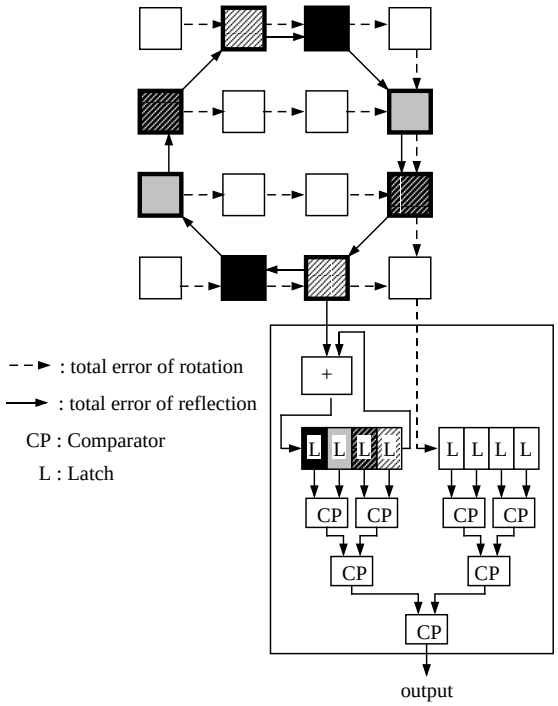

Figure 7. The connection of the calculation of the range block and the rotated and/or reflected domain blocks.

where $I = \lfloor \frac{i}{2} \rfloor$ and $J = \lfloor \frac{j}{2} \rfloor$ 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 such that

$$T_1(d_{I,J}^{d-1}) = \left[T_1(d_{i,j}^d), T_1(d_{i+2,j}^d), T_1(d_{i,j+2}^d), T_1(d_{i+2,j+2}^d)\right],$$
$$T_2(d_{I,J}^{d-1}) = \left[T_2(d_{i+2,j}^d), T_2(d_{i+2,j+2}^d), T_2(d_{i,j}^d), T_2(d_{i,j+2}^d)\right],$$
$$T_3(d_{I,J}^{d-1}) = \left[T_3(d_{i+2,j+2}^d), T_3(d_{i,j+2}^d), T_3(d_{i+2,j}^d), T_3(d_{i,j}^d)\right],$$
$$T_4(d_{I,J}^{d-1}) = \left[T_4(d_{i,j+2}^d), T_4(d_{i,j}^d), T_4(d_{i+2,j+2}^d), T_4(d_{i+2,j}^d)\right],$$
$$T_5(d_{I,J}^{d-1}) = \left[T_5(d_{i+2,j}^d), T_5(d_{i,j}^d), T_5(d_{i+2,j+2}^d), T_5(d_{i,j+2}^d)\right],$$
$$T_6(d_{I,J}^{d-1}) = \left[T_6(d_{i+2,j+2}^d), T_6(d_{i+2,j}^d), T_6(d_{i,j+2}^d), T_6(d_{i,j}^d)\right],$$
$$T_7(d_{I,J}^{d-1}) = \left[T_7(d_{i,j+2}^d), T_7(d_{i+2,j+2}^d), T_7(d_{i,j}^d), T_7(d_{i+2,j}^d)\right],$$
$$T_8(d_{I,J}^{d-1}) = \left[T_8(d_{i,j}^d), T_8(d_{i,j+2}^d), T_8(d_{i+2,j}^d), T_8(d_{i+2,j+2}^d)\right], \quad \text{for } 0 \le I, J \le \frac{N_p}{2} - 2 \text{ and } 0 \le i, j \le N_p - 4, \quad (6)$$

where $T_k, 1 \le k \le 8$ is the isometric transformation function. Note that four subblocks are rotated while each subblock itself is rotated.

The distortion calculation between range block and transformed domain blocks is computed by the summation of the distortion calculation between the d depth range block and the d depth transformed domain blocks. For example, the distortion calculation between $r_{I,J}^{d-1}$ and $T_2(d_{I,J}^{d-1})$ is computed as follows:

$$\begin{aligned}&\delta(r_{I,J}^{d-1}, T_2(d_{I,J}^{d-1})) \\ &\quad = \delta(r_{i,j}^d, T_2(d_{i+2,j}^d)) + \delta(r_{i+1,j}^d, T_2(d_{i+2,j+2}^d)) \\ &\quad +\delta(r_{i,j+1}^d, T_2(d_{i,j}^d)) + \delta(r_{i+1,j+1}^d, T_2(d_{i,j+2}^d)),\end{aligned} \quad (7)$$

where the MAD is used as the distortion criterion.

Equation (8) shows the time at which the domain block $d_{i,j}^d$ is calculated at PE (p, q).

$$\begin{cases} t = P_{pe} - P_{do} & \text{If } P_{pe} \ge P_{do} \\ t = P_{pe} - P_{do} + N_p \times N_p & \text{If } P_{pe} < P_{do} \end{cases}, \quad (8)$$

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 $\mathcal{D}^{d\_max-1}$ with $\mathcal{D}^{d\_max}$ by equations (1)-(8). Each PE is capable of obtaining the information for the current domain block $d_{i,j}^d$ in each PE by equation (8). 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 (6), 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)×(the fast distortion calculation for one domain block)=$(N_p)^2 \times (12C + 3)$.

4.3. 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 $\mathcal{D}^{d\_max}$ and the MADs of all the domain blocks $\mathcal{D}^{d\_max-1}, \ldots, \mathcal{D}^{d\_min}$. Since one block of the upper-level domain pool contains four blocks of the lower-level domain pool, the MADs of upper-level domain pool are obtained by the EA (Error Adder) modules that

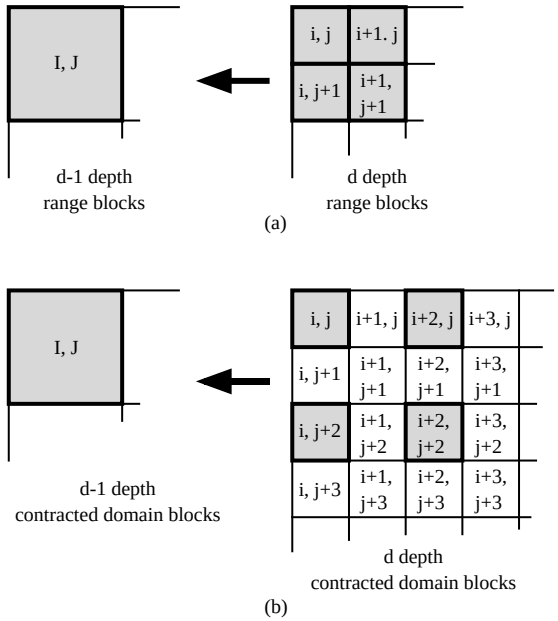

Figure 8. The $d-1$ depth range and domain block consist of four d depth range and domain blocks. (a) range blocks. (b) domain blocks.

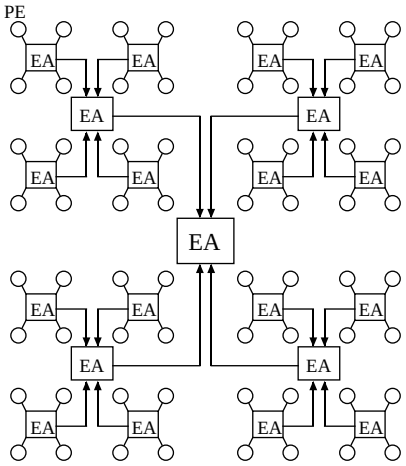

Figure 9. The MAD calculation of other domain pools by the EA modules.

add four MADs of lower-level domain pool (see equation 7 and Figure 9). To obtain the MADs of all the domain pools except $\mathcal{D}^{d\_max}$, $d\_max - d\_min$ steps are needed.

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

The results of the MAD calculation of 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 $\mathcal{D}^{d\_max}$ is the precalculated 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.

5. Conclusions

In this paper, we proposed a parallel architecture for quadtree-based fractal image coding. The quadtree-based encoding scheme is based on the flexible-size range blocks. First, this architecture makes the maximum-depth domain pool and calculates the MADs by using the fast comparison module that is capable of executing eight isometric transformations by one 360 degree rotation around the center. The MADs of other domain pools are computed by adding the MADs of the maximum-depth domain pool. Since the most intensive operation is the MAD calculation of the maximum-depth domain pool, its computational complexity is the same to that of the fixed-size range encoding scheme, that is different from the sequential computing. However, the quadtree-based encoding scheme is capable of yielding better performances than fixed-size range encoding schemes.

References

[1] B. B. Mandelbrot, *The Fractal Geometry of Nature*, Freeman, San Francisco, 1982.

[2] M. F. Barnsley, *Fractals Everywhere*, Academic Press Inc. , San Diego, 1988.

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

[4] A. Jacquin, "Fractal Image Coding: A Review," *Processing of the IEEE,* vol. 81, no. 10, pp. 1451-1465, Oct. 1993.

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

[6] I. K. Kim and R. H. Park, “Still Image Coding Based on Vector Quantization and Fractal Approximation,” *IEEE Trans. Image Processing,* vol. 5, no. 4, pp. 587-597, Apr. 1996.

[7] S. C. Pei, C. C. Tseng, and C. Y. Lin, “A Parallel Decoding Algorithm for IFS Codes Without Transient Behavior,” *IEEE Trans. Image Processing,* vol. 5, no. 3, pp. 411-5, Mar. 1996.

[8] M. S. Lazar and L. T. Bruton, “Fractal Block Coding of Digital Video,” *IEEE Trans. Circuits and Systems for Video Technology,* vol. 4, no. 3, pp. 297-308, June 1994.

[9] Y. Zhao and B. Yuan, “A New Affine Transformation: Its Theory and Application to Image Coding,” *IEEE Trans. Circuits and Systems for Video Technology,* vol. 8, no. 3, pp. 269-274, June 1998.

[10] G. M. Davis, “A Wavelet-Based Analysis of Fractal Image Compression,” *IEEE Trans. Image Processing,* vol. 7, no. 2, pp. 141-154, Feb. 1998.

[11] F. Ancarani, A. De Gloria, M. Olivieri, and C. Stazzone, “Design of an ASIC Architecture for High Speed Fractal Image Compression,” *Proceedings Ninth Annual IEEE international ASIC Conference and Exhibit,* pp. 223-6, 1996.

[12] Z. L. He, M. L. Liou, and K. W. Fu, “VLSI Architecture for Real-Time Fractal Video Encoding,” *1996 IEEE International Symposium on Circuits and Systems,* vol. 2, pp. 738-41, 1996.

[13] O. Fatemi and S. Panchanathan, “Fractal Engine,” *Proc. SPIE Int. Soc. Opt. Eng. ,* vol. 3021, pp. 88-99, 1997.

[14] K. P. Acken, H. N. Kim, K. J. Irwin, and R. M. Owens, “An Architectural Design for Parallel Fractal Compression,” *Proceedings International Conference on Application-Specific System, Architectures and Processors,* pp. 3-11, 1996.

[15] S. Lee and H. Aso, “A Parallel Architecture for High Speed Fractal Image Coding,” *IEEE International Symposium on Parallel Architectures, Algorithms and Networks, Fremantle, Western Australia,* pp. 88-93, July 1999.