設 計 自 動 化 89−10
(1998. 9. 22)

フラクタル画像圧縮の再構成可能アーキテクチャによる実現法

松浦 昭洋　永野 秀尚　名古屋 彰

NTT コミュニケーション科学研究所
〒 619-0237 京都府相楽郡精華町光台 2-4
E-mail: {matsuura, nagano, nagoya}@cslab.kecl.ntt.co.jp

あらまし FPGA 等の再構成可能なハードウェアの出現により，その論理の可変性を活かした情報処理の研究が活発に行われている．本稿では，フラクタル画像圧縮に注目して，対象とする画像に特化したハードウェア構成を取ることで符号化を高速に行う再構成可能アーキテクチャによる実現法を提案する．フラクタル符号化においては画像ブロック間の 2 乗誤差の計算が主要な処理であるが，ハードウェアの再構成が可能な場合，その中で行われる大量の乗算は画像の濃度値を乗数とする定数乗算と見なすことができる．それにより，乗算の実行に汎用乗算器を用いる代わりに，定数に含まれるビットのビットパターンから決まる数のシフトと加減算によって行うことが可能となる．このような構成により，固定的なアーキテクチャによる場合に較べて．ハードウェア上に実現される乗算処理の並列度が 2 倍以上となることを平均的な演算数の見積もりと FPGA を用いた実験により示す．

キーワード フラクタル画像圧縮，IFS，再構成可能アーキテクチャ，FPGA，定数乗算，数表現

A Method for Implementing Fractal Image Compression on Reconfigurable Architecture

Akihiro MATSUURA,　Hidehisa NAGANO　Akira NAGOYA

NTT Communication Science Laboratories
2-4 Hikaridai, Soraku-gun, Seika-cho, Kyoto, 619-0237 JAPAN
E-mail: {matsuura, nagano, nagoya}@cslab.kecl.ntt.co.jp

Abstract By the emergence of reconfigurable hardware such as FPGAs, researches that make use of it have been recently done. In this paper, we focus on fractal image compression and propose a method to implement its encoding phase on reconfigurable architecture. In the encoding, computation of root-mean-squres between two blocks is the most time-consuming part. On reconfigurable architecture, many multiplications in the rms computations can be regarded as constant multiplications. This makes it possible to use shifts and adders instead of multipliers for the computaions. We show by the estimation of the number of operations needed on average and by the experiments that parallelism on the reconfigurable architecture is more than twice than that implemented on fixed architecture.

key words fractal image compression, IFS, reconfigurable architecture, FPGA, constant multiplications, number representations

1 はじめに

FPGA 等の再構成可能なハードウェアの出現により，その内部論理の可変性を活かした情報処理手法に関する研究が活発に行われている．本稿では，我々はデジタル画像の圧縮法の一つであるフラクタル画像圧縮法に注目して，その符号化処理に再構成可能性が効果的に活用できることを示す．フラクタル画像圧縮法 [1]，[2] は，原画像に内在する自己相似性を抽出して，その相似情報を表すアフィン変換のパラメータを用いて画像の符号化を行い，復号はそのアフィン変換を繰り返すことで行う．JPEG にも匹敵する歪み率と圧縮率の高さを有し，エッジの再現性が良く，解像度が広いなどの利点もあることなどから，近年活発に研究が行われている．その符号化，復号化処理のうち，復号化については，アフィン変換を画像の各ブロックに高々 10 数回施すことで復号画像が得られるため処理は比較的軽いが，符号化は相似性の探索回数が膨大で，例えば 256×256 ピクセル程度の画像の符号化に汎用ワークステーションで数十分程度を要するため，その高速化が望まれている．これまで，画像のブロック間のパターンマッチングを行う際の候補となるブロックの数を絞る手法 [3]，[4]，画像ブロックの特徴によって階層構造を作る手法 [5]，超並列計算機によって高速化を計る手法 [6] などが知られている．

今回我々は，フラクタル画像圧縮法で必要となる演算の計算量が画像に依存して異なることに着目し，再構成可能ハードウェアの内部論理の可変性を利用して，画像に適合したハードウェア構成を取って符号化を高速化する，フラクタル画像符号化の再構成可能アーキテクチャによる実現法を提案する．

フラクタル圧縮で最も時間のかかる処理は，ブロック同士の類似度をブロック内の濃度値の 2 乗誤差で計る演算の部分であり，全体の処理の高速化のためにその部分の効率化，並列化は有力である．

2 乗距離計算の並列化では，大量に現れる乗算の処理が問題となるが，我々の構成するネットワークでは，それらの乗算は乗数が画像ブロックのピクセルの濃度値である定数を取る定数乗算と見なせる．ASIC 等の固定的なハードウェアでは，濃度値としてあらゆる場合を想定する必要があることから，これらの乗算のためには汎用の乗算器を置かざるをえないが，ハードウェアの論理の可変性を許せば，定数乗算は正味必要なシフト，加算器のみを用いて実現すればよく，必要なリソース量を大幅に削減することができる．

1 つの濃度値に対応する定数乗算に関する加 (減) 算数の見積もりは，定数を 0，1，-1 等を用いて表現したときに含まれる 0 でない (= 有意な) ビットの数により行うことができる．我々は，2 の補数表現，Canonical Signed Digit (CSD) 表現に対して，平均的にどれだけの加 (減) 算，或いは FPGA 上のロジックエレメント (LE) の数が必要かを評価した．定数乗算のシフト，加算器による実現により，最悪ケースを想定してビット幅分の乗算器を持つ場合に較べて，主要な演算部のリソースに関して，加算器換算で 1/2 から 1/3 に削減できることが分かる．

また，定数内での有意なビットの評価に加え，演算数をより削減するための定数間の演算結果共有手法についても述べる．

最近動的再構成可能なアーキテクチャの研究が盛んになりつつある [7]，[8]．文献 [7] では動画像処理用の動的再構成可能ハードウェアの Testbed に関する報告が行われ，動画の種類により圧縮アルゴリズムを動的に変化させるというアイデアが述べられている．フラクタル圧縮に対する今回の我々のアプローチは，アルゴリズムはフィックスした上で，ハードウェア上に実現される演算器構成を画像に適したものに変えて処理の高速化に重点を置いたものである．

本稿の構成は以下の通りである．2 でフラクタル圧縮法の基本原理について概説し，3 で再構成可能アーキテクチャによる実現法を提案する．4 で実験結果を述べ解析を行う．5 でまとめを述べる．

2 フラクタル画像圧縮法の基本原理

2.1 反復関数系 (Iterated Function System)

ここではフラクタル画像圧縮法として，Barnsley が提案した反復関数系 (以下 IFS)[1] を利用したものを考える．特にここで問題とする濃淡画像の符号化への IFS の応用は Jacquin[2] により行われた．

IFS はジェネレータとなるいくつかのアフィン変換を元に自己相似性を有するフラクタル画像を生成するシステムであり，一方でそれを利用した画像符号化は，画像内の自己相似性を抽出する逆問題に当たる．IFS では，いくつかの縮小アフィン変換からなる組を考え，それらを繰り返し施すことによってフラクタル画像を生成する．それに対して，符号化では画像内の分割された各ブロックについて，相似性を有するより大きなブロックを探し出し，両者の関係を表す縮小アフィン変換のパラメータを保存することで圧縮を行う．

今，$(M,\ d)$ を画像全体からなる距離空間 (画像空間)，d を画像間の距離とする．μ_{orig} を原画像，ν を再生画像として，画像空間の対応を定める写像 $\tau: M \to M$ が縮小写像であるとする．すなわち，ある $s(0 \leq s < 1)$ があって，任意の $\mu_1, \mu_2 \in M$ に対して

$$d(\tau(\mu_1),\ \tau(\mu_2)) \leq s \cdot d(\mu_1,\ \mu_2) \qquad (1)$$

が成り立っているとする．このとき，3角不等式より任意の画像 μ_0 と任意の自然数 n に対して，

$$d(\mu_{orig},\ \tau^n(\mu_0)) \leq \frac{1}{1-s} d(\mu_{orig},\ \tau(\mu_{orig})) + s^n \cdot d(\mu_{orig},\ \mu_0) \qquad (2)$$

が成り立つ．この不等式が意味するところは，もし τ に十分な縮小性があり (s は 0 に近ければ近いほどよい)，$\tau(\mu_{orig})$ が μ_{orig} を十分に近似しているならば，右辺の第二項が十分小さくなるまで τ を繰り返して $\nu = \tau^n(\mu_0)$ とすることで，ν で原画像 μ_{orig} を十分に近似できる，ということである．実際の自然画像の符号化では，原画像を重なりのないブロックに分けて，より大きなブロックで近似していく．次節で符号化アルゴリズムの詳細を述べる．

2.2 IFS を用いた符号化アルゴリズム

本稿で利用する IFS 符号化は，Jacquin[2]，Fisher[9]，[10] によるものである．

まず原画像をレンジブロックと呼ばれる重なりのない正方形のブロックに分割する．それらを近似する候補となるブロック (ドメインブロック) としては，レンジブロックの縦横 2 倍の辺を持つ正方形のブロックを画像中で 1 画素ずつずらしたもの全てを考える．このドメインブロックの集合をドメインプールと呼ぶ．レンジブロック，ドメインブロック共，その画像の複雑度から，シェード，ミッドレンジ，エッジの 3 種類のクラスに分類しておく．

シェード: 濃度に殆んど変化のないブロックのクラス
ミッドレンジ: 濃度差はあるがエッジのないブロック
エッジ: エッジの存在するブロック

シェードに対しては各ピクセルの濃度値を 1 つの定数で置き換え，ミッドレンジ，エッジのクラスに対しては，対応するクラスのレンジブロックとドメインブロックの間で類似度を計る．

ブロック同士の類似度を表す距離を計る際，事前にドメインブロックを 2×2 ピクセルの濃度を平均化するなどして両者のサイズを合わせておく．n をブロックのサイズ，$a_i(1 \leq i \leq n)$，$a_i(1 \leq i \leq n)$ を各々ドメインブロック D_j，レンジブロック R_k のピクセルの濃度値の集合とする．

D_j，R_k 間の距離は，次のように濃度値に関する 2 乗誤差で計る．

$$R(D_j,\ R_k) = \sum_{i=1}^{n} (s \cdot a_i + o - b_i)^2 \qquad (3)$$

```
min_R = threshold_R ;
for (j = 1; j < num_range; j++) {
    for (k = 1; k < num_domain; k++) {
        compute s ;
        if ( 0 ≤ s < 1.2 ) {
            compute R ;
            if ( R < min_R ) {
                min_R = R ;
                also substitute best domain block and s
            }
        }
    }
    if ( min_R == threshold_R ) {
        divide range block R_j into 4 smaller blocks and
        search again ;
    }
}
```

図 1: 最適なドメイン，s，R 等を求めるアルゴリズム

ここで s，o は濃度に関するスケーリング，オフセット成分である．R を s，o に関して偏微分して，R を最小にする s と o を求めると，

$$s = \frac{n\sum_{i=1}^{n} a_i b_i - \sum_{i=1}^{n} a_i \sum_{i=1}^{n} b_i}{n\sum_{i=1}^{n} a_i^2 - (\sum_{i=1}^{n} a_i)^2} \qquad (4)$$

$$o = \frac{\sum_{i=1}^{n} b_i - s\sum_{i=1}^{n} a_i}{n} \qquad (5)$$

このとき R の値は，

$$R(D_j,\ R_k) = \frac{n\sum_{i=1}^{n} b_i^2 - (\sum_{i=1}^{n} b_i)^2}{n} - \frac{n\sum_{i=1}^{n} a_i^2 - (\sum_{i=1}^{n} a_i)^2}{n} s^2. \qquad (6)$$

エッジのクラスに関しては，近似の精度を上げるため，ドメインブロックは元のブロックに加えて，それを 90 度，180 度，270 度回転した 3 種類，及び各々を縦方向の軸に関して反転したものも候補として含める．もしレンジブロックと最も良くマッチしたドメインブロックに関する $R = min\_R$ が閾値 $threshold\_R$ を超える場合は，元のブロックを 4 つの正方形に分割して，同じ大きさ，同じクラスのドメインブロックと再度 s，R の計算を行う．ここではブロックのサイズとして，$n = 8^2 = 64$，$4^2 = 16$ を考えた．

以上の準備の下に，各レンジブロックに対して R の値を最小にする最適なドメインブロックと s，o の値，及び回転，反転の種類を求め，それらの数値に最後に無歪みの圧縮を行うのが符号化の概略である．保存すべき数の有効ビット数としては，経験的に s は 5 ビット，o は 7 ビット程度で良いとされている．アルゴリズムの pseudo code は図 1 のようになる．実行時間という観点

で見ると，R，s 等の計算の反復がその大半を占め，例えば，256 × 256 サイズの一枚の画像の符号化における反復回数をブロックサイズを $8^2 = 64$ として単純に計算すると，32^2(レンジブロックの数)×241^2(ドメインブロックの数) ×8^2(最内部の乗算の数)= 約 38 億回となる．実際この程度の大きさの一枚の画像の符号化には数十分程度を要し，その効率的な処理が重要な課題となっている．

そこで効率化の観点から R，s，o の計算式を見ると次のことが分かる．まず，$\sum_{i=1}^{n} a_i$，$\sum_{i=1}^{n} a_i^2$，$\sum_{i=1}^{n} b_i$，$\sum_{i=1}^{n} b_i^2$ に関しては最も内部のループの中で行う必要はないので，全体の計算量のオーダを上げない限り事前に計算しておく．最内部のループですべきことは，s の値の計算，定数との比較，それから R の値の計算，及び比較の演算である．特に，計算の負荷が大きいのは $\sum_{i=1}^{n} a_i b_i$ を計算する部分である．

なお復号化は，画像の各レンジブロックに随伴する s，o 等アフィン変換のパラメータの値とレンジブロックに対応するドメインブロックを用い，一度の反復の中で，対応するドメインブロックの濃度値を s 倍して，o を加えた値をそのレンジブロックの濃度値とする処理を全てのレンジブロックに対して行う．任意の初期画像からこの処理の反復を十数回繰り返せば，近似画像に収束していく．

3 再構成可能アーキテクチャによる実現法

3.1 再構成可能アーキテクチャ上のネットワーク構成

ASIC 等の固定的なハードウェアの場合，各演算器に関して，入力の最悪ケースのリソース量を用意しておく必要があるが，その部分をハードウェアの論理の可変性を活かして実現すれば，必ずしも最悪ケースを想定したリソースを用意する必要がなくなる．ここでは，再構成可能なハードウェア上で，実現される PE のサイズを各々が分担する画像ブロックの濃度値によって可変にして，限定されたリソース上で実現される PE の数も変化する実現法を考える．

基本的な方針としては，各レンジブロックに 1 つの PE を割り当て，PE は，ドメインブロックの濃度情報を入力として $\sum_{i=1}^{n} a_i b_i$，s，R などの計算をパイプライン処理で行っていく．以下，ネットワーク構成について説明する．

符号化では，ドメインプールのブロックのうち，2.2 で述べたミッドレンジ，エッジの 2 種類のクラス毎間で類似度を計る計算を行うため，ドメインプールの各々のクラスのブロックは，対応するレンジブロックの PE のネットワークに対して入力される．PE 間の接続は隣接

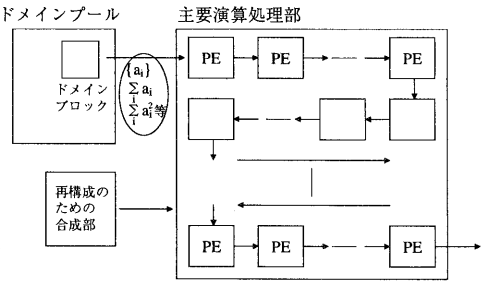

図 2: 符号化のためのネットワーク構成 (ミッドレンジの場合)

する PE 同士で行い，ドメインブロックのデータ $\{a_i\}$，$\sum_{i=1}^{n} a_i$，$n\sum_{i=1}^{n} a_i^2 - (\sum_{i=1}^{n} a_i)^2$ 等が 1 クロック毎に絶え間なく入力され，隣接する PE 間の接続によってドメインブロックの情報が転送される．ミッドレンジクラスのブロックの処理のためのネットワークの構成の概略は図 2 の通りである．エッジの場合は，ドメインブロックに回転，反転を施して，同様の PE のネットワークに 8 回入力される．レンジブロックに対応する全ての PE がハードウェア上に載らない場合は，ハードウェア上の PE，及びネットワークを順次再構成し，全てのレンジブロックの符号化が終わるまで処理を行う．

以下，各 PE の内部構成，処理内容について説明していく．

3.2 PE の構成

PE 内のハードウェアの構成に関して，次のような二つのフェーズに分ける．まず $\sum_{i=1}^{n} a_i b_i$ の計算部を P_1 とし，それ以下の計算の部分を P_2 と呼ぶ．P_2 部では (4)，(6) 式に従って，s, R 等を計算し，min_R の値を更新する．

ブロックサイズを n とすると，一括して入力される 1 ドメインブロック内のデータ数には 1 から n まで任意性がある．P_2 は入力数の単位によらないが，P_1 部分は，一度に入力されるデータ数が変われば構成も変化する．

$n = 64$ とし，ドメインブロックの 64 個の濃度値を一括入力するとして，P_1 部分を木構造で実現した際のデータフローグラフが図 3 である．

P_1 の構造としては同様の木構造のものを使うと仮定する．入力データ数として最適なものがいくつかを考える．入力データ数 k と $2k$ のときを較べたとき，サイズ

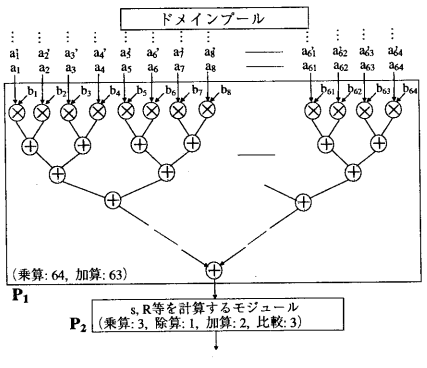

図 3: PE

k のPE2個による並列処理でサイズ $2k$ のPE1個分の処理を行い，サイズ k のPE2個において，P_2 部分は同じリソースをパイプライン処理で共有できるので，いくつかのPEがハードウェアに同時に実現されるような場合に両者の処理時間，必要なリソース量に大差はない．したがって，我々はPEのサイズは出来る限り大きなものを用い，n 個の定数を一括して入力することとする．

レンジブロックをサイズ n で作った場合，PEには，以下のようにハードウェアの再構成可能性を活かした実現法が考えられる．濃度値を $2^8 = 256$ 階調とすると，固定的なハードウェアでは，P_1 部分は8ビットの乗算器，及び16ビット以上で必要なビット幅の加算器を使うのが最も自然な実現法である．しかし，P_1 の最初の n 個の乗算の部分では，乗数である $\{b_i\}_{i=1}^{n}$ はそのPEの持つ定数成分なので，ハードウェアの再構成が許されれば，必ずしも汎用乗算器を用意する必要はない．すなわち，正味 $\{b_i\}$ に関する演算を実現するだけの演算器を用意すればよく，より具体的には必要最低限のシフトと加減算を行うためのリソースを用意すればよい．

乗数が定数であることは，$\sum b_i$ 等を乗数とする乗算についても言え，P_2 の中でこれらに関する乗算も定数乗算で行うことでリソースを削減できる．

また，2.2で述べたように例えば s の値の有効ビット数は5であるので，各演算を実現する演算器もその有効桁を残すだけの演算を行えば良い．

次節では，乗算器で固定的に作られる部分を定数乗算に置き換えた場合の処理手法について述べる．

3.3 定数乗算における演算数削減手法

本節では，まずハードウェアが再構成可能か固定的かが符号化時間，特に主要演算部である P_1 部の乗算の処理時間にどのような違いとして現れるかを調べ，続いて数表現に基づいた演算削減手法，複数の定数間で演算結果を共有する手法について述べる．シフトと加減算のコストにおいては，シフトを配線で実現すれば加減算のコストが時間的にも面積的にも大きくなるため，ここでは，特に加減算の削減を考えることとする．

3.3.1 再構成可能なハードウェアで演算を行う効果

3.2で述べた n 定数乗算を一度に計算する P_1 に関して，演算回数削減の効果は次のように見積もることができる．まずレンジブロックの集合 $\Psi = \{\psi_i\}_{i=0}^{m}$ を処理する際の各レンジブロックの処理に要するリソース量を考える．ハードウェアが固定的な場合は，各乗算を実現する演算器は想定されるあらゆるレンジブロックの濃度値に対応できるものでなければならない．つまり，濃度値を $2^8 = 256$ 階調とすれば，一般には n 個の8ビット乗算器を用意することになってしまう．P_1 の乗算部に8ビット乗算器を用いた場合，P_1 の演算リソース量は各ブロックで一様に

$$H_{mult} \tag{7}$$

となる．一方，再構成可能なハードウェアを考えると，PEのサイズはレンジブロックの濃度値によって可変である．レンジブロック ψ_i の処理に要する P_1 部のリソース量を H_{ψ_i} とすると，m 個のレンジブロックの処理に要する延べリソース量は $\sum_{i=1}^{m} H_{\psi_i}$ である．したがって，レンジブロック一個当たりに要する平均リソース量は，

$$\frac{\sum_{i=1}^{m} H_{\psi_i}}{m} \tag{8}$$

となる．これらの演算器をハードウェア上に実現して処理を行う場合，(7)，(8)のリソース量は，ハードウェア上の処理の並列度に直接反映される．すなわち，PE一つ当たりに必要なハードウェアリソース量が削減できれば，それだけ多くのPEを同時にハードウェア上に実現して符号化時間を削減することができる．極端な例としては，レンジブロックの全てのピクセルの濃度値が，2進数で表したときにただ一つのビットしか立っていないような場合，つまり $10\cdots00$ のような数では，P_1 の最初の乗算部ですべきことは単にシフトのみであり，乗算器を $n(=64)$ 個用意しなければならない場合とのリソース量の違いは大きい．

また，乗算を定数乗算で行う場合のメリットとして，レンジブロックの濃度値 $\{b_i\}$ のためのメモリを各PEに用意する必要がない，ということもあげられる．現状のFPGA等の再構成可能なデバイスでは，リソースをメモ

リとして使うと集積度が低く，メモリ要素をPEに持たせることは，結果的に同時に載せることのできるPEの数を減らしてしまう．その意味でもメモリを使わず布線論理で実行できる演算手法は有効である．

実際上は，処理時間の見積もりにおいて，さらに利用するデバイスの種類，集積度，及びハードウェアの再構成の時間が重要な要素である．それらについては今後の課題であるが，以上の議論は，ビットレベルで評価した必要リソース量が入力によって変化するような問題に対して，再構成可能アーキテクチャによる処理が有望であることを示唆していると考えられる．

以下，レンジブロックに対するリソース量H_{ψ_i}の中で，特にP_1部分の見積もりについて検討していく．

3.3.2 定数乗算による回路規模の削減

(1) 数表現に基づく手法

まず数の表現として2の補数表現を考える．この場合，定数におけるビット1の数は乗算を行う場合の加減算数に直接影響する．つまり，もし演算共有のアルゴリズムを全く用いなければ，ある定数を乗数として別の数との乗算を行う際の加算回数は(定数内のビット1の数)-1である．今ビット0と1が一様に現れる条件で考えると，2の補数表現では，ビット幅をNとすると(但し符号ビットを除く)，平均的に$N/2$の0でないビットを持つので，加算数は平均的に$N/2-1$回必要である．

さらに，2の補数表現よりも0でない(=有意な)ビットの数が少ない表現も存在する．

CSD(Canonical Signed Digit) 表現

CSD表現は0，$+1$，-1を用いた数の表現方法であり，2つ以上の連続するビットが0でないという特徴を持つ．例えば，$39 = 100111_2$(2進数)はCSD表現では$10100(-1)$と表される．CSD表現における0でないビットの数に関して，次の結果が知られている([11])．

Proposition. 半開区間$[-1, 1)$内のNビットのCSD表現された数における0でないビットの数の平均値は$N/3+1/9+O(2^{-N})$である．

すなわち，CSD表現された数は0でないビットを平均的に約$N/3$個持ち，演算回数も$N/3-1$回程度とできる．これは前章で述べたP_1の乗算部分のリソース量を加減算換算で約1/3にできることを意味している．PEの後半部P_2はP_1と較べて小さいため，CSD表現された数による定数乗算を行うと，P_1で汎用乗算器を用いる場合と較べて，全体でも2倍以上の並列化と速度向上が見込める．

(2) 部分演算の結果の共有に基づく手法

次に複数の定数乗算の間で演算結果を共有する手法，及びその際のアルゴリズム選択における注意点について述べる．定数乗算をシフトと加減算で行いコストを削減する手法が，ディジタルフィルタや行列積等に対して提案されている[12]，[13]が，ここでは$\sum_{i=1}^{n} a_i b_i$の計算への応用を考える．

Example.: $\{a_i\}$を変数組，$\{b_i\}$を次のような定数組として，$\sum_{i=1}^{4} a_i b_i$の計算を加算とシフトで計算する:
$b_1 = 1111_2$，$b_2 = 1100_2$，$b_3 = 0011_2$，$b_4 = 0001_2$.
$\sum_{i=1}^{4} a_i b_i$の計算に各ビット1に関してシフト，加算を順次繰り返す手法を取ると，

$$\sum_{i=1}^{4} a_i b_i = \{a_1 + (a_1 \ll 1) + (a_1 \ll 2) + (a_1 \ll 3)\} + \{(a_2 \ll 2) + (a_2 \ll 3)\} + \{a_3 + (a_3 \ll 1)\} + a_4 \quad (9)$$

となり加算8回，シフト6回必要だが，次のようにすればさらに演算数が削減できる:

$$\sum_{i=1}^{4} a_i b_i = (a_1 + a_3) + (a_1 + a_3) \ll 1 + (a_1 + a_2) \ll 2 + (a_1 + a_2) \ll 3 + a_4 \quad (10)$$

$a_1 + a_2$，$a_3 + a_4$等は一度計算しておけばよいので加算6回，シフト3回でできる．探索を徹底的にすればそれだけ加算数も削減されるが，その探索時間が全体の処理時間に負担になってはならないので，この部分には符号化のアルゴリズムより計算量の軽いアルゴリズムを選ばざるをえない．図4は2数ずつのペアのみで演算結果の共有を計る手法を表している．これは，2つの定数$(b_i,\ b_{i+1})$(iは奇数)において，1が共に存在する桁が複数あれば，それは$a_i + a_{i+1}$という計算が複数存在することを意味するので，そのビットのペアを優先的に処理するアルゴリズムである．2数のペアという部分は定数個の数の組を一度に考えることも可能である．

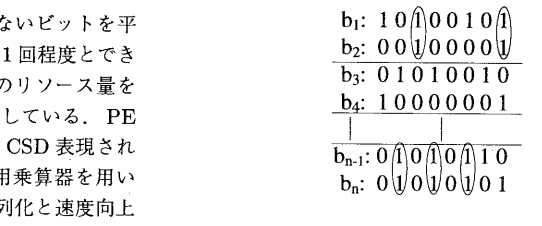

図4: ビットのペアの探索

2の補数表現に基づいて，定数間の演算共有を行わずに $\sum_{i=1}^{n} a_i b_i$ を計算した場合と上記の削減手法を用いた場合のデータフローグラフ生成の手間を考えると，前者は $\{b_i\}_{i=1}^{n}$ の各ビットが1かどうかを判断して，順次シフト，加算していく場合を考えればよく，$O(nN)$ の手間で出来るが，後者は，2数のペア $(b_i,\ b_{i+1})$(i は奇数)に関して N 桁分 $(1,\ 1)$ というビットのペアがあるかどうかを判断すればよく，2数のペアが $n/2$ 個あるので，増える手間は，$O(n/2 \cdot N) = O(nN)$ となり，オーダ的にも負担はかからない．2進表現でビット0と1が一様に現れる条件で考えると，N ビットの2数で同じ桁が共に1となっているものは，N 桁のうち平均的に $N/4$ 桁あり，2数間の加算結果 $a_i + a_{i+1}$ (i は奇数) の共有によって加減算数を $N/4-1$ 回削減できる．したがって，PE の P_1 全体で $(N/4-1) \times n/2$ 回削減できる．$N = 8$，$n = 64$ の場合だと平均32加算の削減となる．

定数乗算で演算共有を行った場合の，乗算器を n 個使用する場合に対する P_1 部分の加算数の削減率は，

$$\frac{(N/2 \times n) + (N/4-1) \times n/2}{(N-1)n + (n-1)} = \frac{5}{8} - \frac{1}{2N} + O(\frac{1}{nN})$$

となる．$n = 64$，$N = 8$ のときの削減率は $288/(512-1) =$ 約 $9/16$ である．

なお，ここで述べた部分演算共有アルゴリズムは PE の高位合成，或いは論理合成で考慮されることになる．ハードウェアの合成においては多くの困難な最適化問題を解くこととなり，今回述べたアルゴリズムのオーダに関するのと同様の問題が現れる．すなわち，合成の各フェーズでどのようなアルゴリズムを用いるかが問題となる．再構成可能なハードウェア上で，合成と問題の解決を並行して行うことで問題を解く場合，合成時間が実際上も理論上も処理の高速化のネックになる可能性がある．(動的) 再構成可能アーキテクチャの有用性を増すために，その問題への対策が課題である．

4 実験結果と考察

ここでは，現在入手可能なデバイス，マッピングツールを用いて PE の P_1 部分を実現した際のリソース量と3.3.2で取り上げた定数乗算による加算数の見積もりを比較し，さらに，定数乗算を利用した場合の演算数を SIDBA の画像 "Girl"(256×256 pxls, 8b/p)，及び "Lenna"(480×512 pxls, 8b/p) を用いて評価する．

実験は，FPGA のデバイスは Altera の FLEX10KA Family，マッピングツールは MAX+PLUS II (ver. 8.3) を用いた．ブロックサイズ $n = 4^2 = 16$，$8^2 = 64$ に対して P_1 の木構造と P_2 を実現した．今回，乗算器としては Wallace tree 乗算器を用い，加算器では演算の各段でのビット幅も考慮し，桁上げ先見加算器をベースに構成した，今回の FPGA への実装と LE 数の評価でレジスタは考慮していない．ビット幅による必要ロジックエレメント (LE) 数は表1の通りである．

	LE数
8ビット乗算器	163
8ビット加算器	14
16ビット加算器	35
20ビット加算器	46

表 1: 演算器毎の LE 数

なお，2進数ではビット0と1が同等に現れると仮定した．また乗算を定数乗算で行う場合，ブロックサイズによらず，加算器のビット幅は8ビットから15ビットまで変化するが，表1から，1加算当たりの平均 LE 数を25個と仮定し，P_1 で $a_i b_i$ を木構造部分で加算する部分では，ブロックサイズが64のときは加算器のビット幅は16から22，ブロックサイズが16のときはビット幅は16から20となるが，両サイズ共ビット幅の期待値は17ビット以下であるため，木の中の加算器の LE 数は全て35個と仮定した．

P_1 部分に要したリソースを乗算器ベースと定数乗算を用いた場合で比較すると，表2のようになる．乗算器ベースで乗算器の占める LE の割合は8割強である．乗算器ベースの場合と2進表現を元に定数乗算の場合の LE 数を較べると，乗算器ベースに対して 定数乗算を用

(乗算器ベース)

サイズ	乗算	加算	総LE数
4 x 4	16	15	3094
8 x 8	64	63	12517

(定数乗算)

サイズ	乗算	加算	総LE数
4 x 4	0	63	1725
8 x 8	0	255	7005

表 2: P_1 部の演算数，LE 数

サイズ	乗算	除算	加算	比較	総LE数
4 x 4	3	1	2	3	1343
8 x 8	3	1	2	3	1556

表 3: P_2 部の演算数，LE 数

画像	乗算器ベース	定数乗算	
		2進表現	CSD表現
Girl	524287	264651 (50.5%)	211610 (40.4%)
Lenna	1966079	947632 (48.2%)	800035 (40.7%)

表 4: 画像 Girl, Lenna の P_1 部の延べ加減算数

いる場合はサイズが 16, 64 のとき共に約 44% 削減された. LE 数削減に関して, 定数乗算を用いる効果が良く現れている.

一方, P_2 の演算部 (乗算 3 回, 除算 1 回, 加算 2 回, 比較 3 回) の LE 数に関する結果は表 3 のようになった. ブロックサイズにより LE 数が異なるのは, サイズが大きくなると, 計算途中に現れる数のビット幅が大きくなるためである. 演算数が少ないにもかかわらず LE 数が多いのは, P_2 部では P_1 部に較べて各演算のビット幅が大きいのが要因と考えられる. 定数乗算の部分, s の有効桁が固定で 5 桁である点などは回路規模削減のために利用した.

最後に "Girl", "Lenna" における濃度値の値を 2 進表現, または CSD 表現した際の加減算回数を示す (表 4). 両画像共, 濃度値を数表現した際の 0 でないビットの数から求められる加減算数は, 汎用の乗算器の演算回数を加算数で見積もった場合に較べて, 2 進表現では約 50%, CSD 表現では約 60% 削減された. これにより, 実際の画像に対しても定数乗算の有効性が確認できた.

5 まとめ

フラクタル画像符号化の再構成可能アーキテクチャによる実現法を提案した. 再構成可能アーキテクチャでは, フラクタル画像符号化の主要な処理である乗算を画像毎に定まる濃度に関する定数乗算と見なしたハードウェア構成を取ることができ, それにより, 入力画像の各ブロックに特化した演算器構成を取ることが可能となる. これは ASIC 等の固定的なアーキテクチャにおいて, 考えられる全ての入力を想定した最悪ケースのリソースの演算器を作る必要があるのに対する優位点であると考えられる. 実験により, 定数乗算を用いた場合, 汎用乗算器を作る場合に較べて演算数が半数以下となり, FPGA 上に実現した際の LE 数も演算数に応じて削減することが確認できた.

これからの課題としては, 再構成可能アーキテクチャが優位性を持つ問題のクラスの拡大, ハードウェアの合成時間に関する詳細な検討, このようなアーキテクチャ向きのデバイスの検討などが挙げられる.

参考文献

[1] M. F. Barnsley, "Fractals Everywhere", AK Peters (1993).

[2] A. E. Jacquin, "Fractal Image Coding: A Review", Proc. IEEE, Vol. 81, No. 10, Oct., pp. 1451-1465 (1993).

[3] G. E. Oien, S. Lepsoy, and T. A. Ramstad, "An Inner Product Space Approach to Image Coding by Contractive Transformations", Proc. ICASSP'91, pp. 2773-2776 (1991).

[4] D. M. Monro and F. Dudbridge, "Fractal Approximation of Image Blocks", Proc. ICASSP'92, pp. 485-488 (1992).

[5] 中野 勝彦, 中川 匡弘, "ガウシアンピラミッドを利用した階層的 IFS 画像符号化", 信学会論文誌 A, Vol. J-78-A, No. 7, pp. 856-863 (1995).

[6] 曽根原 登, "並列コンピュータによるニューラルネットワーク処理", 電気学会, LAV-91-9, pp. 37-47, (Aug., 1991).

[7] 関山 守, 野村 真義, 平木 敬, "Hardware での動画処理における動的再構成の優位性について", 情処研報, 98-ARC-130-23, pp. 133-137 (Aug., 1998).

[8] 小栗 清, 伊藤 秀之, 小西 隆介, 名古屋 彰, "布線論理による汎用計算機構 (プラスティックセルアーキテクチャ)", 信学技報, CPSY-98-54, pp. 45-52 (Aug., 1998).

[9] Y. Fisher, "Fractal Image Compression", SIGGRAPH'92 Course Notes (1992).

[10] Y. Fisher (Ed.), "Fractal Image Compression, Theory and Application", Springer (1992).

[11] R. I. Hartley and K. K. Parhi, "Digit-Serial Computation", Kluwer Academic Publishers (1995).

[12] M. Mehendale, S. D. Shelekar, and G. Venkatesh, "Synthesis of multiplier-less FIR filters with minimum number of additions," Proc. ICCAD'95, pp. 668-671 (1995).

[13] A. Matsuura, M. Yukishita, and A. Nagoya, "A Hierarchical Clustering Method for the Multiple Constant Multiplication Problem", IEICE Trans. on Fund., Vol. E80-A, No. 10, Oct., pp.1767-1773 (1997).