社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

書換え可能な二変数関数の数値計算回路について

永山　忍[†]　笹尾　勤[††]　Jon T. Butler[†††]

† 広島市立大学大学院 情報工学専攻　〒 731–3194 広島市 安佐南区 大塚東 3–4–1
†† 九州工業大学 電子情報工学科　〒 820–8502 福岡県 飯塚市 川津 680–4
††† 海軍大学院大学　アメリカ カリフォルニア州 モントレー

あらまし　本稿は, 二変数数学関数を計算する数値計算回路の設計法および書換え可能な回路構成を提案する. 提案手法では, 二変数関数を実現するために, 与えられた定義域 (平面) を複数の小さな領域に分割し, 各領域毎に関数を多項式で近似する. 二変数関数の定義域を効率よく分割するために, 本稿では, 再帰的平面分割アルゴリズムを提案する. また, 本稿では, 多様な二変数関数を実現できる二種類の書換え可能な回路構成を示す. 提案する回路構成により, 二変数関数回路のシステマチックな設計が可能になる. FPGA を用いた実験により, 本数値計算回路は, 一変数関数回路の組合わせで設計された二変数関数回路に比べ, 58% のメモリ量かつ 39% の遅延時間で関数を実現できることを示す.
キーワード　数値計算回路, 二変数関数, 区分多項式近似, 再帰的平面分割アルゴリズム, 書換え可能な回路構成.

On Programmable Two-Variable Numerical Function Generators

Shinobu NAGAYAMA[†], Tsutomu SASAO[††], and Jon T. BUTLER[†††]

† Department of Computer and Network Engineering, Hiroshima City University, Ozuka-Higashi 3–4–1, Asa-Minami-Ku, Hiroshima, 731–3194 Japan
†† Department of Computer Science and Electronics, Kyushu Institute of Technology, Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
††† Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA 93943-5121 USA

Abstract　This paper proposes a design method and programmable architectures for numerical function generators (NFGs) of two-variable functions. To realize a two-variable function in hardware, we partition a given domain of the given function into segments, and approximate the function by a polynomial in each segment. This paper introduces two planar segmentation algorithms that efficiently partition a domain of a two-variable function. This paper also introduces two architectures that can realize a wide range of two-variable functions. Our architectures allow a systematic design of two-variable functions. FPGA implementation results show that, for a complicated function, our NFG achieves 58% of memory size and 39% of delay time of a circuit designed using one-variable NFGs.
Key words　Numerical function generators (NFGs), two-variable functions, piecewise polynomial approximation, recursive planar segmentation algorithm, programmable architecture.

1. はじめに

数学関数は, コンピュータグラフィックスやディジタル信号処理などの様々な分野で, 加算や乗算と同様に基本的な演算として広く利用されており, 高性能 CPU の 1 ～ 2 桁高速に数学関数を計算する様々な回路の設計法が提案されている [11]. しかし, ほとんどの既存手法 [4, 9, 13–16] が一変数数学関数のみを対象としており, 多変数 (二変数以上の) 数学関数を対象とした回路の設計法は, あまり報告されていない [5, 6, 17]. 文献 [5, 6, 17] で報告されている手法は, 特定の関数だけを対象とした専用回路の設計法なので, 関数毎に異なる設計法が必要になってしまう. 私たちの知る限り, 任意の多変数関数のシステマチックな設計法は未だ報告されていない.

任意の多変数関数の単純な設計法として, 関数表を単一メモリで直接実装する方法がある. この手法は, 関数値をメモリに記憶するだけなので, 任意の関数を容易に実装できる. また, 一回のメモリアクセスだけで関数を計算できるため, 非常に高速である. しかし, m 変数関数を n ビット (各入力変数が n ビット) で実装するのに, 2^{mn} ワードのメモリが必要になるため, m と n が小さいときでも, この手法は実用的でない.

実用的な実装を得るために, 多変数関数は, 一変数関数回路, 加算器, および乗算器の組合せでしばしば設計される [5, 6]. この設計法は, 実装に必要なメモリ量を削減できるが, 関数によっては, 複雑な回路構成のため低速な回路をもたらしてしまう場合がある. また, 複雑な回路構成では, 誤差解析が難しくなり, 結果として回路の出力精度の保証が困難になってしまう.

本稿では, 与えられた二変数関数を直接設計するシステマチックな手法を提案する. 本提案手法は, 関数の区分多項式近似に基づいているので, 複雑な関数に対しても, 回路構成は単純なままである. 与えられた二変数関数を区分多項式で効率よく近似す

るために, 本稿では, 二つの平面分割アルゴリズムを提案する. 提案するアルゴリズムは, 与えられた二変数関数の定義域を効率よく分割できることを示す. また, 本稿では, 多様な二変数関数を実現できる二種類の書き換え可能な回路構成も提案する.

本稿の構成は, 以下の通りである. 第 2. 節で, 本稿で使用する数値表現および決定グラフを定義する. 第 3. 節で, 二つの平面分割アルゴリズムを提案し, 第 4. 節で, 二変数関数のための二種類の回路構成を提案する. そして, 第 5. 節で, 提案アルゴリズムと回路構成を実験的に評価し, 第 6. 節で, 本稿の内容をまとめる. 本数値計算回路の誤差解析は, [12, 15] とほぼ同様であるため, ページ数削減のために割愛する.

2. 用語の定義

2.1 数値表現と誤差

［定義 1］ **二進固定小数点**で表現された数値 X を

$$X = (x_{l-1}\, x_{l-2}\, \ldots\, x_1\, x_0.\, x_{-1}\, x_{-2}\, \ldots\, x_{-m})$$

と表記する. ただし, $x_i \in \{0,1\}$, l は整数部のビット数, m は小数部のビット数である. 本稿では, 負数を 2 の補数で表現する.

［定義 2］ **誤差**とは, もとの値との差分絶対値を意味し, 特に, 本稿では, 関数近似で生じる誤差を**近似誤差**, 値を有限桁の二進固定小数点で表現した際に生じる誤差を**丸め誤差**という. **許容誤差**とは, 仕様で許容できる誤差の最大値を意味し, 特に, **許容近似誤差**は, 許容できる近似誤差の最大値を意味する.

［定義 3］ **確度 (accuracy)** は, 実数計算における小数点以下の有効桁数を意味する. m **ビット確度**とは, 実数計算において小数点以下の有効桁数が m ビットであることを意味する. 特に, 最大誤差が 2^{-m} のときの m ビット確度を **1 ulp** (unit in the last place) という [11]. 本稿で, m ビット確度の数値計算回路は, 小数部 m ビット入力, 小数部 m ビット出力でなおかつ, 出力値が 1 ulp の回路を意味する. 1 ulp の出力値を得るためには, 計算途中では, それ以上の確度で計算する必要がある.

2.2 決定グラフ

［定義 4］ 二分決定グラフ (BDD: Binary Decision Diagram) [2, 10] は, 論理関数: $\{0,1\}^n \to \{0,1\}$ を根付き有向グラフで表現したもので, 論理関数 f にシャノン展開 $f = \overline{x_i} f_0 + x_i f_1$ を繰り返し適用することで得られる. BDD は, 論理関数値 0 および 1 を表現する二つの終端節点と, 入力変数を表現する非終端節点から成る. 各非終端節点は, 変数値に対応する重みなしの二つの枝, 0-枝と 1-枝を持つ.

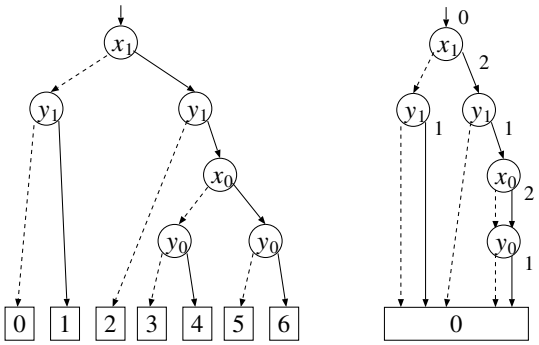
［定義 5］ MTBDD (Multi-Terminal BDD) [3] は, BDD を拡張した表現法であり, 整数関数: $\{0,1\}^n \to Z$ を表現する. ただし, Z は整数集合を表す. MTBDD には, 整数値を表現する複数の終端節点がある.

［定義 6］ EVBDD (Edge-Valued BDD) [7, 8] も, BDD を拡張した表現法であり, 整数関数を表現する. EVBDD は, 整数関数 f に展開 $f = \overline{x_i} f_0 + x_i(f_1' + \alpha)$ を繰り返し適用することで得られる. ここで, $f_1 = f_1' + \alpha$, α は, f_1 の定数項を意味する. EVBDD は, 0 を表現する一つの終端節点と, 重み付きの 1-枝を持った非終端節点から成る. EVBDD で, 各 1-枝には, 整数値の重み α が割り付けられ, 0-枝には, 重み 0 が割り付けられている.

［例 1］ 図 1(b) の MTBDD および (c) の EVBDD は, 図 1(a) で与えられる関数 f を表現している. 図 1(b) と (c) で, 破線と実線は, それぞれ 0-枝と 1-枝を表している. EVBDD の 1-枝には整数値の重みが割り付けられている. MTBDD では, 入力変数の値に従ってグラフを辿り, 到達した終端節点の値から関数値を得ることができる. 一方, EVBDD では, 入力変数の値に従い, 終端節点までグラフを辿ったときの枝の重みの総和から関数値を得る. (例終り)

x_1	y_1	x_0	y_0	f	x_1	y_1	x_0	y_0	f
0	0	0	0	0	1	0	0	0	2
0	0	0	1	0	1	0	0	1	2
0	0	1	0	0	1	0	1	0	2
0	0	1	1	0	1	0	1	1	2
0	1	0	0	1	1	1	0	0	3
0	1	0	1	1	1	1	0	1	4
0	1	1	0	1	1	1	1	0	5
0	1	1	1	1	1	1	1	1	6

(a) 関数表.



(b) MTBDD (c) EVBDD

図 1 整数関数を表現する MTBDD と EVBDD

3. 平面分割に基づく区分多項式近似

3.1 平面分割問題

与えられた二変数関数を区分多項式で近似するためには, 与えられた関数の定義域を小さな領域に分割する必要がある. 二変数関数の定義域は, 平面で形成されているので, 平面分割アルゴリズムが必要になる. 区分多項式近似に基づく数値計算回路のメモリ量や速度は, 分割アルゴリズムの効率に大きく左右されるので, 高速かつコンパクトな数値計算回路を設計するためには, 効率の良い平面分割アルゴリズムが重要になる. ここで効率の良い平面分割とは, 以下を満足するような分割である.

（1） 関数近似に必要な領域数が少ない.

（2） 平面分割を実現する回路の複雑度が小さい.

領域数は, 数値計算回路のメモリ量に直接影響を与えるため, 少ないほど望ましいが, 平面分割を実現する回路の複雑さも同時に考慮する必要がある. 仮に領域数最小の分割が見つかったとしても, 分割が複雑な場合, 分割を実現する回路が大きくなり, 結果として, 大きく低速な数値計算回路を生成してしまう. 特に, 平面分割の実現は, 線形分割の実現 [12] よりも遙かに複雑なので, 実現する回路が大きくなりやすい. そのため, これらの二つを同時に考慮した平面分割アルゴリズムが, 高速かつコンパクトな数値計算回路の設計に不可欠である. また, 設計時間を短縮するために, 分割アルゴリズム自体の複雑さも考慮することが重要である. そこで, 本稿では, 二つの発見的な平面分割アルゴリズムを提案する.

3.2 平面分割アルゴリズム

本節では, まず, 領域数と分割の複雑さの両方の削減を考慮した**再帰的平面分割アルゴリズム** (図 2) について説明する. このアルゴリズムは, 入力として, 二変数関数 $f(X,Y)$, X と Y の定義域 $\{[X_b, X_e), [Y_b, Y_e)\}$, X と Y の確度 m_{in}, 近似多項式の次数 d, および許容近似誤差 ε_a を与えると, 各領域の最大近似

入力:	二変数関数 $f(X,Y)$, X と Y の定義域 $\{[X_b,X_e),[Y_b,Y_e)\}$, X と Y の確度 m_{in}*, 近似多項式の次数 d, 許容近似誤差 ε_a. *X と Y のビット数は同じ.
出力:	領域 $\{[X_b,P_0),[Y_b,Q_0)\},\{[X_b,P_0),[Q_0,Q_1)\}\ldots,\{[P_{r-1},X_e),[Q_{r-1},Y_e)\}$, 補正値 $v_0,v_1,\ldots,v_{k-1}$.
処理:	
1.	領域 $\{[X_b,X_e),[Y_b,Y_e)\}$ における近似多項式 $g_d(X,Y)$ を求める.
2.	正の最大誤差 $max_{fg}=\max\{f(X,Y)-g_d(X,Y)\}$ を計算する.
3.	負の最大誤差 $min_{fg}=\min\{f(X,Y)-g_d(X,Y)\}$ を計算する.
4.	近似誤差を $\varepsilon_d=(max_{fg}-min_{fg})/2$ とし, 近似多項式の補正値を $v=(max_{fg}+min_{fg})/2$ とする.
5.	$\varepsilon_d<\varepsilon_a$ または $(X_e-X_b)\leqq 2^{-m_{in}}$ ならば, 再帰処理を終了する.
6.	そうでなければ, $P=(X_b+X_e)/2$, $Q=(Y_b+Y_e)/2$ とし, 領域 $\{[X_b,X_e),[Y_b,Y_e)\}$ を 4 つの等しい領域 $\{[X_b,P),[Y_b,Q)\}$, $\{[X_b,P),[Q,Y_e)\}$, $\{[P,X_e),[Y_b,Q)\}$, $\{[P,X_e),[Q,Y_e)\}$ に分割する.
7.	各領域に対し, 最大近似誤差が ε_a より小さくなるまで, 処理 1, 2, . . . , 6 を再帰的に繰り返す.

図 2　再帰的平面分割アルゴリズム.

誤差が許容近似誤差より小さくなるまで, 領域の四等分割を再帰的に繰り返し, 最終的に定義域の平面分割を得る. このアルゴリズムで生成される領域は, 大きさが $w_i\times w_i$ の正方形に制限されることに注意. ここで, $w_i=2^{h_i}\times 2^{-m_{in}}$, h_i は, 領域毎に異なる整数である. すなわち, 分割点 P_i と Q_i は, 下位 h_i ビットが 0 の点 (つまり, $P_i=(\ldots\ p_{-j+1}\ p_{-j}\ 00\ \ldots\ 0)$, ここで $j=m_{in}-h_i$) に制限される. 図 2 で, 近似多項式 $g_d(X,Y)$ は, 領域の中心 (u,v) における $f(X,Y)$ のテイラー展開:

$$g_d(X,Y)=f(u,v)+\left(s\frac{\partial}{\partial X}+t\frac{\partial}{\partial Y}\right)f(u,v)$$
$$+\left(s\frac{\partial}{\partial X}+t\frac{\partial}{\partial Y}\right)^2\frac{f(u,v)}{2!}+\ldots+\left(s\frac{\partial}{\partial X}+t\frac{\partial}{\partial Y}\right)^d\frac{f(u,v)}{d!}$$

で得られる. ここで, $s=X-u, t=Y-v, u=(B_x+E_x)/2$, $v=(B_y+E_y)/2$. 近似多項式 $g_d(X,Y)$ の最大近似誤差を削減するために, 補正値 v を用いて $g_d(X,Y)$ を垂直方向に平行移動することで, 正の最大誤差と負の最大誤差を等しくしている. そのため, 最大近似誤差は, $(max_{fg}-min_{fg})/2$ になり, $g_d(X,Y)+v$ が数値計算回路で実装される近似多項式になる.

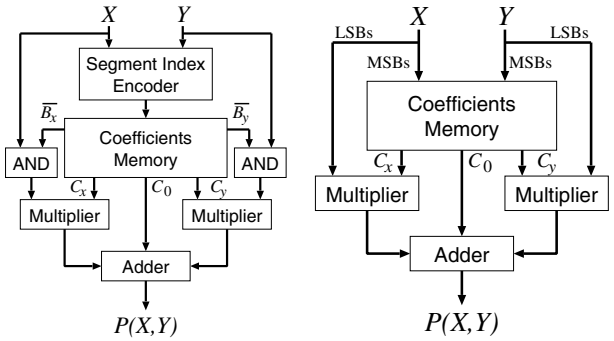
次に, **等領域平面分割アルゴリズム**について説明する. 再帰的平面分割アルゴリズムでは, 得られる領域が不均等であるため, 入力変数 X と Y の値に対応する領域を検索するために, 分割を実現する回路が必要になるが, 等領域平面分割アルゴリズムで得られる領域は, X と Y の上位ビットにそのまま対応しているので, 分割を実現する回路が不要になる. そのため, このアルゴリズムを使うことで, より高速な数値計算回路の生成が期待できる. 等領域分割を得るために, まず, 図 2 のアルゴリズムを用いて, 与えられた許容近似誤差で関数を近似するために必要な最小の領域を見つける. そして, その最小の領域で, 定義域を等領域に分割する.

4. 二変数関数数値計算回路の構成

4.1 再帰的分割と等領域分割に基づく回路構成

本回路構成では, 平面分割アルゴリズムで生成された各領域 $\{[B_x,E_x),[B_y,E_y)\}$ において, 関数 $f(X,Y)$ をそれぞれ異なる多項式 $P(X,Y)$ で近似する. 前節で述べたように, 近似多項式 $P(X,Y)$ は, 関数 f のテイラー展開に補正値 v_i を加えたものである. この多項式を展開し, X と Y について整理することで, 以下の式に変形できる.

$$P(X,Y)=C_0+C_x(X-B_x)+C_y(Y-B_y)\qquad(1)$$
$$+C_{xy}(X-B_x)(Y-B_y)+C_{x2}(X-B_x)^2$$
$$+C_{y2}(Y-B_y)^2+\ldots+C_{yd}(Y-B_y)^d$$



(a) 再帰的分割に基づく回路構成. (b) 等領域分割に基づく回路構成.

図 3　平面近似による二種類の二変数関数回路の構成.

図 3 は, 平面近似による二種類の二変数関数回路の構成を示している. これらの回路構成は, 区分平面多項式近似 ((1) の最初の三つの項) のみに対応しているが, 高次多項式への拡張は容易である. 図 3(a) と (b) は, 再帰的分割に基づく回路構成と等領域分割に基づく回路構成をそれぞれ示している. 図 3(a) の領域指定回路 (Segment Index Encoder) は, X と Y を入力として, 対応する領域の領域番号を出力する. そして, この領域番号を係数表 (Coefficients Memory) のアドレスとして使用し, 対応する近似多項式の係数を読み出す. 読み出した係数を使って, 乗算器と加算器で近似値 $P(X,Y)$ を計算する. また, 再帰的分割では, [13] で示されているように, $X-B_x$ と $Y-B_y$ の計算を $\overline{B_x}$ および $\overline{B_y}$ との AND 演算で実現できるため, 図 3(a) では, $X-B_x$ と $Y-B_y$ の計算を AND ゲートで実現していることに注意されたい.

一方, 等領域分割では, 領域番号と $X-B_x$ および $Y-B_y$ の値が, それぞれ, X と Y の上位ビットと下位ビットから直接得られるので, 図 3(b) に示されているように, 領域指定回路も AND ゲートも必要ない.

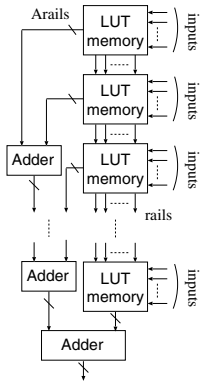
近年の FPGA には, 論理素子だけでなく, 同期メモリブロックや専用乗算器が備わっているので, これらの回路構成は, FPGA 上で効率よく実装することができる.

4.2 領域指定回路 (Segment Index Encoder) の構成と設計法

領域数を k とし, X と Y がそれぞれ n ビットで表現されているとすると, 領域指定回路は, 図 4(a) に示されている領域指定関数: $\{0,1\}^n\times\{0,1\}^n\to\{0,1,\ldots,k-1\}$ を実現する回路である. 本提案手法では, 図 4(b) に示されている回路構成で実現する. この構成で, 隣接する LUT 間の信号線をレール (rail) といい, レールは, 後に示すように, EVBDD を辿るときの部分グラフを表現する. そして, LUT から加算器へ出力される信号線を A

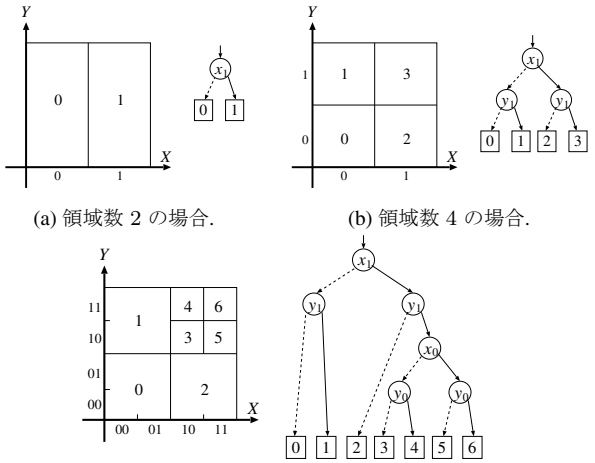
領域	領域番号
$X_b \leqq X < P_0$ $Y_b \leqq Y < Q_0$	0
$X_b \leqq X < P_0$ $Q_0 \leqq X < Q_1$	1
⋮	⋮
$P_{r-1} \leqq X < Y_e$ $Q_{r-1} \leqq X < Y_e$	$k-1$

(a) 領域指定関数.



(b) LUT と加算器による実現 [13].

図 4　領域指定回路.



(a) 領域数 2 の場合.

(b) 領域数 4 の場合.

(c) 領域数 7 の場合.

図 5　再帰的平面分割 (領域指定関数) を表現する MTBDD.

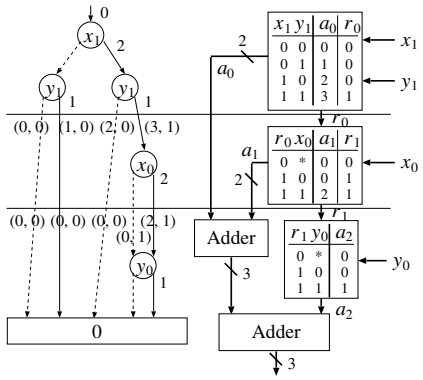


図 6　EVBDD の分解と領域指定回路.

レール (Arail) とよび, A レールは, EVBDD を辿ったときの枝の重みの和を表現する. 以下に, 図 2 のアルゴリズムで生成された平面分割を, どのように図 4(b) の回路構成で実現するかを示す.

まず, 図 5 に示されているように, 図 2 の再帰的平面分割アルゴリズムで得られた領域指定関数を MTBDD で表現する. そして, 得られた MTBDD を EVBDD に変換し, EVBDD を図 6 のように分解することにより, 図 4(b) の回路構成を得ることができる. 図 6 で, 各 LUT 内の表中の 'r_i' は EVBDD の部分グラフを表現するレールを表し, 'a_i' は枝の重みの和を表現する A レールを表す. EVBDD で, 横線と交わる枝に割り当てた "(a_i, r_i)" は, 枝の重みの和と部分グラフをそれぞれ表す. この回路構成の詳細については, 文献 [13] を参照されたい.

この回路構成で, 再帰的分割を実現する LUT のサイズは, 以下の定理で示すように, 領域数に依存する.

［定理 1］ 再帰的平面分割で得られる領域指定関数は, レール数および A レール数が高々$\lceil \log_2 k \rceil$ の領域指定回路 (図 4(b)) で実現できる. ここで, k は領域数を表す.(注1)

このため, 本数値計算回路では, 領域数の削減は, 係数表だけでなく領域指定回路のサイズ削減にも有効である.

本数値計算回路の係数表と領域指定回路の LUT は, FPGA 上の組込 RAM(例えば, アルテラ FPGA の M4K など) で実装されるので, RAM の中身を書き換えることで, 様々な二変数関数を単一の回路構成で実現できる. RAM データを書き換えるだけなので, 本数値計算回路は, FPGA を再構成することなく実現する二変数関数を変更できるという特長を持つ.

(注1)：証明は, ページ数制限のために割愛する.

5. 実験結果

5.1 領域数と平面分割アルゴリズムの計算時間

表 1 は, 文献 [1] 内の様々な二変数関数に, 3. 節で提案した二つの平面分割アルゴリズムを適用したときの領域数とその計算時間を示している. これらの結果は, 二変数関数を平面 (一次) 多項式で近似するのに必要な領域数と計算時間である. 表中で, $WaveRings$, $Gaussian$, $Beta$ は, 以下のように定義される関数である.

$$WaveRings = \frac{\cos\left(\sqrt{X^2+Y^2}\right)}{\sqrt{X^2+Y^2+0.25}} \quad Gaussian = \frac{1}{Y\sqrt{2\pi}} e^{-\frac{X^2}{2Y^2}}$$

$$Beta = 2\int_0^{\frac{\pi}{2}} \sin^{2X-1}\theta \, \cos^{2Y-1}\theta \, d\theta = \frac{\Gamma(X)\Gamma(Y)}{\Gamma(X+Y)}$$

表 1 から, $\sin(\pi XY)$ を除いた全ての関数において, 再帰的分割法が等領域分割法に比べ大幅に少ない領域数で関数を近似できることがわかる. 特に, より高い確度において, 再帰的分割に必要な領域数は, 等領域分割に必要な領域数の数パーセントだけになる. $\sin(\pi XY)$ に関しては, 12 ビット確度においても, 等領域分割と再帰的分割の領域数の差がそれほど大きくない. この結果から, この関数は, 等領域分割に適しており, 等領域分割でもコンパクトな数値計算回路を生成できることがわかる. また, 表 1 の結果から, 提案した二つのアルゴリズムはどちらも短い時間で平面分割を実行できることがわかる. このような高速な分割は, 数値計算回路の設計時間を短縮するに有効である.

5.2 二変数関数回路に必要なメモリ量

表 2 は, 図 3 に示した二種類の二変数関数回路のメモリ量を比較している. 再帰的分割に基づく二変数関数回路は, 係数表と領域指定回路 (LUT) の二種類のメモリを必要とするので, これらに必要なメモリ量の和が表で示されている.

表 2 から, 再帰的分割に基づく二変数関数回路は, 領域指定回路を使用しているにも関わらず, 全ての関数において, 等領域分割に基づく回路よりも遙かに少ないメモリ量で関数を実現していることがわかる. 例えば, 再帰的分割に基づく 12 ビット確度の二変数関数回路は, 等領域分割に基づく回路のわずか 0.6% のメモリ量で, $XY/\sqrt{X^2+Y^2}$ を実現している.

メモリ量と確度の関係を調べるために, 様々な確度で $XY/\sqrt{X^2+Y^2}$ の回路を設計し, 図 7 のようなグラフを得た. この図で, 縦軸は対数目盛で表現されていることに注意. 図 7 は, 以下の三種類の回路のメモリ量を比較している.

（1） $f(X,Y)$ の関数表を実現する単一メモリ

表 1　平面近似に基づく二つの平面分割法における領域数の比較.

No.	二変数関数 $f(X,Y)$	定義域	X と Y が 8 ビット確度の場合 (許容近似誤差: 2^{-10})					X と Y が 12 ビット確度の場合 (許容近似誤差: 2^{-14})				
			領域数		R_s	計算時間 [秒]		領域数		R_s	計算時間 [秒]	
			再帰的	等領域	[%]	再帰的	等領域	再帰的	等領域	[%]	再帰的	等領域
f_0	$\sin(\pi X)\ln(Y)$	$0 \leqq X < 1, 0 < Y < 1$	4,696	65,280	7	0.19	0.02	244,807	16,773,120	1	8.9	7.2
f_1	$\sin(\pi X)\sqrt{Y}$	$0 \leqq X < 1, 0 < Y < 1$	1,393	16,384	9	0.07	0.61	38,773	16,773,120	0.2	1.7	6.7
f_2	$\sin(\pi XY)$	$0 \leqq X < 1, 0 \leqq Y < 1$	1,486	4,096	36	0.07	0.19	26,122	65,536	40	1.2	3.2
f_3	X^4Y^5	$0 \leqq X < 1, 0 \leqq Y < 1$	457	4,096	11	0.03	0.20	8,179	262,144	3	0.5	11.1
f_4	$1/\sqrt{X^2+Y^2}$	$0 < X < 1, 0 < Y < 1$	3,835	65,025	6	0.11	0.01	173,552	16,769,025	1	5.0	5.1
f_5	$XY/\sqrt{X^2+Y^2}$	$0 < X < 1, 0 < Y < 1$	376	4,096	9	0.01	0.11	6,523	1,048,576	0.6	0.2	22.5
f_6	$WaveRings$	$0 \leqq X \leqq \pi, 0 \leqq Y \leqq \pi$	1,619	10,201	16	0.08	0.39	28,377	646,416	4	1.3	18.9
f_7	$Gaussian$	$0 < X < 1, 0 < Y < 1$	3,182	65,025	5	0.12	0.02	141,113	16,769,025	0.8	5.5	7.1
f_8	$\sqrt{X^2+Y^2}$	$0 < X < 1, 0 < Y < 1$	355	4,096	9	0.01	0.12	6,160	1,048,576	0.6	0.2	24.6
f_9	$\sqrt[3]{X^3+Y^3}$	$0 < X < 1, 0 < Y < 1$	400	16,384	2	0.04	0.76	6,790	4,194,304	0.2	0.5	188.8
f_{10}	$Beta$	$1/8 \leqq X < 1, 1/8 \leqq Y < 1$	5,815	50,176	12	0.73	0.05	187,201	3,211,264	6	24.6	326.5

再帰的: 再帰的分割.　　等領域: 等領域分割.　　R_s: 再帰的 / 等領域 × 100 (%).

計算時間: 各アルゴリズムが分割に要する CPU 時間.

< 実験環境 >　CPU: Intel Xeon 2.6GHz　メモリ: 1GB　OS: Redhat Linux　C コンパイラ: gcc -O2

表 2　二種類の二変数関数回路に必要なメモリ量.

No.	8 ビット確度の数値計算回路			12 ビット確度の数値計算回路		
	再帰的	等領域	R_m	再帰的	等領域	R_m
f_0	260,052	783,360	33	16,683,510	285,143,040	6
f_1	59,511	360,448	17	2,030,356	201,277,440	1
f_2	69,352	110,592	63	1,313,684	2,293,760	57
f_3	25,392	102,400	25	516,230	8,126,464	6
f_4	226,403	1,040,400	22	13,054,030	402,456,600	3
f_5	18,120	90,112	20	369,189	27,262,976	1
f_6	100,030	346,834	29	1,886,924	23,917,392	8
f_7	186,980	910,350	21	11,345,482	368,918,550	3
f_8	16,882	94,208	18	316,128	28,311,552	1
f_9	21,792	278,528	8	405,576	88,080,384	0.5
f_{10}	291,735	602,112	48	11,814,069	122,028,032	10

R_m: 再帰的 / 等領域 × 100 (%).

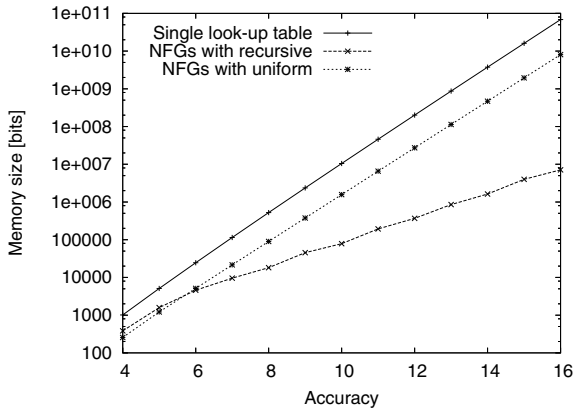


図 7　$XY/\sqrt{X^2+Y^2}$ における, メモリ量と確度の関係.

（2）　再帰的分割に基づく二変数関数回路

（3）　等領域分割に基づく二変数関数回路

興味深いことに, この関数では, 等領域分割に基づく回路のメモリ量は単一メモリと同じように増加していく. 一方, 再帰的分割に基づく回路のメモリ量は, 他の二つに比べ, 遙かにゆっくりと増加していく. 16 ビット確度において, 再帰的分割に基づく回路のメモリ量は, 等領域分割に基づく回路のわずか 0.09% だった.

表 3　8 ビット確度の二変数関数回路の FPGA 実装結果.

FPGA デバイス:	Altera Stratix (EP1S10F484C7)									
論理合成ツール:	Synplify Pro Ver. 8.8									
No.	再帰的分割に基づく回路					等領域分割に基づく回路				
	LE	DSP	周波数 [MHz]	段数	遅延	LE	DSP	周波数 [MHz]	段数	遅延
f_0	347	4	149	9	60	–	0	–	1	–
f_1	206	2	149	7	47	58	1	183	3	16
f_2	280	2	169	7	41	62	2	183	3	16
f_3	280	2	169	7	41	57	2	183	3	16
f_4	552	4	169	9	53	–	0	–	1	–
f_5	180	2	169	6	35	56	2	183	3	16
f_6	364	4	149	8	54	78	2	183	3	16
f_7	430	4	149	10	67	–	0	–	1	–
f_8	177	2	169	6	35	60	2	183	3	16
f_9	189	2	169	6	35	56	0	343	3	9
f_{10}	439	4	169	9	53	–	0	–	1	–

表中の “–” は組込 RAM が不足し実装できなかったことを示す.

LE: 論理素子数.　DSP: 使用された DSP(乗算器) 数.

周波数: 動作周波数.　段数: パイプライン段数.

遅延: 回路の入力から出力までに要する時間 [nsec.].

5.3　FPGA 実装結果

表 3 は, 二種類の構成に基づく 8 ビット確度の二変数関数回路を Altera Stratix FPGA (EP1S10F484C7) に実装した結果を比較している.

等領域分割に基づく二変数関数回路は, 領域指定回路がないため, 再帰的分割に基づく回路よりパイプライン段数が短く, 遅延時間が短くなる. しかし, 実装には多くのメモリ量が必要なため, FPGA への実装が困難になる. 表 3 で, パイプライン段数が 1 の関数は, 等領域の数が多すぎるため, 係数表のサイズが関数表のサイズと等しくなり, 結果的に単一メモリで実現されている. 実験に用いた FPGA では, メモリ不足のため, これら 4 つの関数を実装することができなかった. 一方, 再帰的分割に基づく

表 4　一変数関数回路の組合せで設計した二変数関数の FPGA 実装結果.

FPGA デバイス: Altera Stratix (EP1S10F484C7)						
論理合成ツール: Synplify Pro Ver. 8.8						
二変数関数 $f(X,Y)$	メモリ量 [bits]	LE	DSP	周波数 [MHz]	段数	遅延時間 [nsec.]
$\sin(\pi X)\ln(Y)$	7,104	234	4	149	7	47
$XY/\sqrt{X^2+Y^2}$	31,104	381	13	133	12	90
$WaveRings$	15,232	410	10	149	11	74

メモリ量: 二変数関数の実現に必要な総メモリ量.

二変数関数回路は, 実験に用いた全ての関数を高い動作周波数で FPGA に実装できた.

二変数関数は, しばしば一変数関数回路と加算や乗算などの基本演算の組合せで実現される. 例えば, $\sin(\pi X)\ln(Y)$ は, $\sin(\pi X)$ と $\ln(Y)$ をそれぞれ実現する二つの一変数関数回路とそれらの積を取る乗算器で実現できる. そこで, 二変数関数を直接設計する本提案手法と, 一変数関数回路の組合せを用いる従来手法との比較を行った.

比較実験を行うために, 以下の三つの二変数関数を一変数関数回路の組合せを用いる従来手法で手設計し, 表 3 と同じ FPGA に実装した.

（1） $\sin(\pi X)\ln(Y)$
（2） $XY/\sqrt{X^2+Y^2}$
（3） $WaveRings$

表 3 の二変数関数回路と同じ確度を保証するために, 関数毎に誤差解析を行った. また, 各一変数関数回路の設計には, 線形近似と不等区間分割に基づく手法 [13] を用いた. 表 4 に, 一変数関数回路の組合せで実装した結果を示す.

表 3 と表 4 の比較から, $\sin(\pi X)\ln(Y)$ を除いて, 提案手法は, 従来手法より少ない論理素子数と乗算器数で, なおかつより短い遅延時間で二変数関数を実現できることがわかる. $XY/\sqrt{X^2+Y^2}$ と $WaveRings$ において, 本提案回路の遅延時間は, それぞれ従来手法で設計された回路の 39% と 73% に削減されている. 特に, $XY/\sqrt{X^2+Y^2}$ においては, 表 2 から, メモリ量も 58% まで削減されていることがわかる.

以上の結果から, 二変数関数の設計に一変数関数回路用いることで, 必要なメモリ量を大幅に削減できることがわかる. しかし, 関数によっては, そのような設計法では, 複雑な回路構成のため, 遅い回路が生成されてしまう場合がある. また, 複雑な回路構成は, 誤差解析を複雑にし, 回路の出力精度の保証が困難になる. そのため, このような手法は, 多くの設計時間を要する.

6. 結論とコメント

本稿では, 二変数関数を計算する数値計算回路の設計法と書き換え可能な回路構成を提案した. 二変数関数をハードウェアで実現するために, 本手法では, 与えられた定義域を小さな領域に分割し, 各領域毎に関数を多項式で近似する. 本稿では, 与えられた定義域を効率よく分割するための二つの平面分割アルゴリズムを提案した. 私たちの知る限り, 本提案手法が区分多項式近似を用いた最初の二変数関数のシステマチックな設計法である. 実験により, 本提案手法は, 従来手法に基づいて手設計された二変数関数回路より高性能な回路を自動生成できることを示した.

本稿で提案されたアルゴリズムや回路構成を, 三変数以上の関数に拡張することは容易である.

謝　　辞

本研究の一部は, 平成 20 年度科学研究費補助金 (若手研究 (B)) 課題番号 20700051 および平成 20 年度広島市立大学教員特定研究費 (一般研究) 課題番号 8108 による.

文　　献

[1] H. Anton, *Multivariable Calculus*, John Wiley & Sons, Inc., 1995.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[3] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 54–60, June 1993.

[4] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," *16th IEEE Inter. Conf. on Application-Specific Systems, Architectures, and Processors (ASAP'05)*, pp. 328–333, 2005.

[5] R. Gutierrez and J. Valls, "Implementation on FPGA of a LUT based $atan(y/x)$ operator suitable for synchronization algorithms," *Proc. of the IEEE Conf. on Field Programmable Logic and Applications*, pp. 472–475, Aug. 2007.

[6] Z. Huang and M. D. Ercegovac, "FPGA implementation of pipelined on-line scheme for 3-D vector normalization," *Proc. of the 9th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'01)*, pp. 61–70, Apr. 2001.

[7] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," *Proc. of 29th ACM/IEEE Design Automation Conference*, pp. 608–613, 1992.

[8] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "EVBDD-based algorithms for linear integer programming, spectral transformation and functional decomposition," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 8, pp. 959–975, Aug. 1994.

[9] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Hierarchical segmentation schemes for function evaluation," *Proc. of the IEEE Conf. on Field-Programmable Technology*, Tokyo, Japan, pp. 92–99, Dec. 2003.

[10] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD – Foundations and Applications*, Springer, 1998.

[11] J.-M. Muller, *Elementary Function: Algorithms and Implementation*, Birkhauser Boston, Inc., New York, NY, second edition, 2006.

[12] S. Nagayama, T. Sasao, and J. T. Butler, "Compact numerical function generators based on quadratic approximation: Architecture and synthesis method," *IEICE Trans. Fundamentals*, Vol. E89-A, No. 12, pp. 3510–3518, Dec. 2006.

[13] S. Nagayama, T. Sasao, and J. T. Butler, "Design method for numerical function generators using recursive segmentation and EVBDDs," *IEICE Trans. Fundamentals*, Vol. E90-A, No. 12, pp. 2752–2761, Dec. 2007.

[14] J.-A. Piñeiro, S. F. Oberman, J.-M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. on Comp.*, Vol. 54, No. 3, pp. 304–318, Mar. 2005.

[15] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE Transactions on Computers*, Vol. 56, No. 6, pp. 826–838, Jun. 2007.

[16] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Trans. on Comp.*, Vol. 48, No. 8, pp. 842–847, Aug. 1999.

[17] N. Takagi and S. Kuwahara, "A VLSI algorithm for computing the Euclidean norm of a 3D vector," *IEEE Transactions on Computers*, Vol. 49, No. 10, pp. 1074–1082, Oct. 2000.