Vol.2011-SLDM-150 No.8
2011/5/19

一般化並列カウンタを用いたマルチオペランド加算器合成問題のILPによる定式化

松永　多苗子[†1]　木村晋二[†2]　松永裕介[†3]

FPGA 上でマルチオペランド加算を実現する場合，桁上げを伝播させずにオペランド数を圧縮する回路を一般化並列カウンタを用いて LUT 上に実現する方が，キャリーチェインを用いた加算木構成よりも，高速に実現できることが最近の研究で分かってきた．本稿では，オペランド数圧縮回路を構築する過程を，用いる GPC の種類と個数を決定する過程とそれに基づいて回路を構築する過程に分割し，前半部分を整数線形計画問題として定式化し段数および要素数の最小化を行う手法を提案する．

ILP-based Multi-Operand Adder Synthesis on FPGAs using Generalized Parallel Counters

Taeko Matsunaga,[†1] Shinji Kiura[†2]
and Yusuke Matsunaga[†3]

Recent researches suggest that multi-operand adders can be efficiently realized on FPGAs by using compressor trees which reduce the number operands without any carry propagation. This paper addresses compressor tree synthesis based on Generalized Parallel Counters (GPCs). The target problem is regarded as the minimization problem of the total number of GPCs under the minimum level of GPCs, and an ILP-based algorithm is proposed.

†1 早稲田大学 IT 研究機構
Information Technology Research Organization, Waseda University
†2 早稲田大学大学院情報生産システム研究科
Graduate School of Information, Production and Systems, Waseda University
†3 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electorical Engineering, Graduate School of Kyushu University

1. はじめに

マルチオペランド加算は 3 個以上の入力データの和を計算する演算で，乗算や積和演算等，より複雑な演算で必要となる基礎的な算術演算である．高速なマルチオペランド加算を ASIC で実現する場合，まず桁上げを伝播させずにオペランド数を 2 個にまで減らした上で，最後に 1 回だけ桁上げ伝播加算器（Carry-Propagate Adder, CPA）を用いる方法が一般的に用いられている．オペランド数の圧縮は，全加算器や半加算器を構成要素として，Wallace tree や Dadda tree 等の構成で実現することが可能である[1)–4)]．

一方，FPGA の場合には，圧縮回路を LUT 上に実現するより，デバイスに特有のキャリーチェイン構造を用いた加算木として実現する方が高品質な結果が得られると考えられていた．しかし，LUT の入力数の増加に伴い，全加算器よりも入力から出力への削減率の大きい一般化並列カウンタ（Generalized Parallel Counter, GPC）を用いることにより，圧縮回路を効率よく LUT 上に実現可能であることが示唆され，いくつかのヒューリスティック[5)–8)] や Integer Linear Programming (ILP) を用いたアプローチ[9)] が提案されている．

圧縮回路の遅延および面積は，圧縮回路を構成する GPC の段数と要素数で粗く見積もることが可能である．また，圧縮回路の動的消費電力も GPC の要素数と段数に影響をうけることが観測されており[8)]，要素数と段数を最小化することが，遅延，面積，消費電力のどの指標に対しても有効であるといえる．しかし，既存のヒューリスティックは段数，要素数ともに最小性を保証するものではなく，ILP による手法も段数の最小化を目標としているが，要素数に関しては間接的に減少させるのみであった．

本稿では，GPC を構成要素とするオペランド数圧縮回路の，段数および要素数の最小化問題を対象とし，ILP によるアプローチを提案する．本手法の特徴は，既存の ILP 手法のように回路構造に対応した変数を導入するのではなく，オペランド数を圧縮していくフローに対応した定式化を行うこと，および，段数と要素数の両方の最小化を目指すことである．スケーラビリティの点で問題があると予想されるが，小規模な回路に対してでも段数および要素数の最小解を求めることは，既存のヒューリスティックの評価やよりよいヒューリスティックを模索する上でも有意義であると考えられる．

以下，2 節で圧縮回路合成の概要といくつかの定義を行った上で，3 節において ILP による定式化について説明する．4，5 節で既存手法との比較，及び，実験結果を示し，6 節でまとめを行う．

Vol.2011-SLDM-150 No.8
2011/5/19

2. GPC を構成要素とするオペランド数圧縮回路合成

本稿では近年の FPGA デバイスによく見られるように LUT の入力数は 6 であるとする．また，3 入力加算を高速に実行できる構造が備わっていると仮定し，オペランド数は 3 個まで圧縮するものとする．

2.1 オペランド数圧縮回路合成の概要

以下では，マルチオペランド加算器合成問題の入力となるオペランドは 1 ビットごとにドットで表し，加算すべきすべてのオペランドをドットのマトリックスの形で表現する．ここで，マトリックスの各列は対応するビット位置を表している．このマトリックスをドットダイアグラムと呼ぶ．各列のドット数をその列の高さと呼び，すべての列の中で最大の高さを，そのドットダイアグラムの高さと呼ぶ．

一般的なオペランド数圧縮回路合成の流れを図 1 に示す．図上部はマルチオペランド加算の例として，乗算と，同じビット幅のオペランドの加算のドットダイアグラムを示している．与えられたドットダイアグラムに対して，想定する基本構成要素を用いて高さの圧縮が繰り返し行われ，使用した構成要素が適切に接続されて圧縮回路が構成される．

2.2 圧縮回路の構成要素

Wallace や Dadda 方式の乗算器の場合，圧縮回路の構成要素は全加算器と半加算器であるが，これらは単一列に対する並列カウンタの一種とみなすことができる．

定義 1 単一列に対する並列カウンタ $(m;n)$ とは，m 個のビットを入力し，1 の数を数えて，その個数を n ビットの符号なし整数として出力する組み合わせ回路である．出力は 0 から m の範囲の値であり，m と n の間には関係 $n = \lceil log_2(m+1) \rceil$ が成り立つ．

$A = (a_{n-1}a_{k-2}\dots a_0)$ を n ビット符号なし 2 進整数とする．a の添字 i を，a_i のランクと呼ぶとすると，$(m;n)$ カウンタは，同じランク i の入力の加算を行い，出力のランクは $i, i+1, \dots, i+(n-1)$ となる．(3;2) カウンタは全加算器，(2;2) カウンタは半加算器と同等である．

一般化並列カウンタ GPC は，単一列に対する並列カウンタを拡張したのである．

定義 2 一般化並列カウンタ（Generalized Parallel Counter,GPC）とは，異なるランクのビットを入力し，それらの和を計算して n ビット符号なし整数として出力する組み合わせ回路である．ランク i の入力の個数を m_i とすると，GPC は $(m_{k-1}, m_{k-2}, \dots, m_0; n), m_{k-1} > 0$

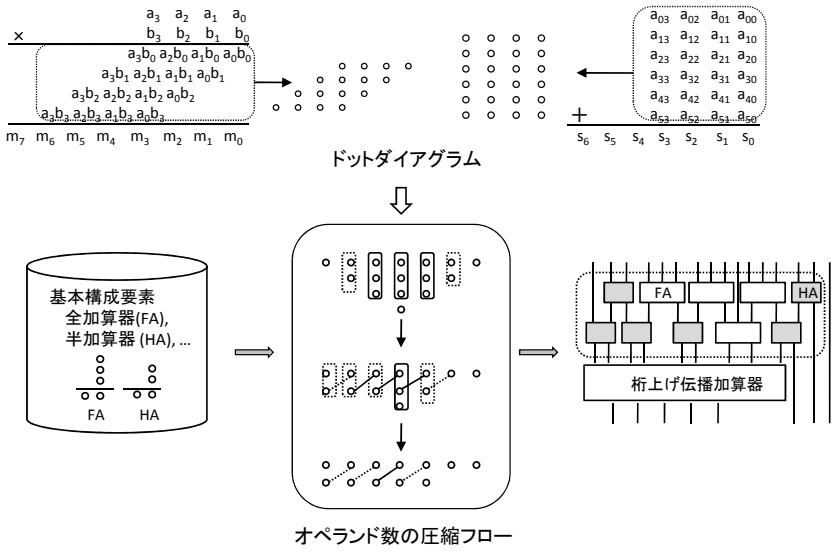


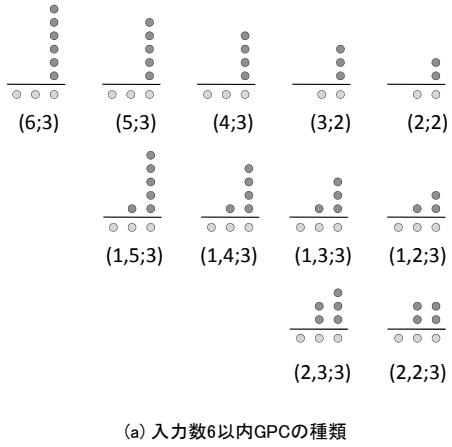
図 1 オペランド数圧縮回路合成の概要
Fig. 1 Overview of compressor tree synthesis

で表される．出力の値は 0 から M の範囲の値であり，M と n の間には

$$M = \sum_{i=0}^{k-1} m_i 2^i \quad , n = \lceil \log_2(M+1) \rceil$$

が成り立つ．入力の総数を $m = \sum_{i=0}^{k-1} m_i$ とで表すとすると，GPC の総入力数 m の総出力数 n に対する比 $R = \frac{m}{n}$ は，この GPC の削減率と呼ばれる．

オペランド数を圧縮するためのカウンタも，ドットダイアグラムを用いて表すことができる．図 2 (a) に入力数 6 以下としたときの，可能な GPC の種類を示す．ここで，横線の上の部分が入力ドット，下の部分が出力ドットを表す．(b) は，最下位ビットのランクを j として，各 GPC の各ランクにおけるドット数の増減を表している．例えば，(1,5;3) GPC



(a) 入力数6以内GPCの種類

(b) GPC適用後の各ランクのドット数の増減

GPC	j+2	j+1	j
(6;3)	+1	+1	-5
(5;3)	+1	+1	-4
(4;3)	+1	+1	-3
(3;2)		+1	-2
(2;2)		+1	-1
(1,5;3)	+1	0	-4
(1,4;3)	+1	0	-3
(1,3;3)	+1	0	-2
(1,2;3)	+1	0	-1
(2,3;3)	+1	-1	-2
(2,2;3)	+1	-1	-1

図 2 入力数 6 以下の GPC 集合
Fig. 2 The set of GPCs with at most 6 inputs

は，ランク j ではドット数が 5 個から 1 個に減るので，全体として 4 個減少することを表している．

2.3 圧縮回路の面積，遅延，消費電力の指標

GPC を構成要素とする圧縮回路の実現は，基本的に 1 つの GPC は 1 出力あたり 1 個の LUT で実現できることを前提としている．LUT での実現コストが各 GPC で一定であるとすると，対象デバイスを特定しないレベルでの面積，遅延は，GPC の総数，最大段数で粗く見積もることが可能である．消費電力に関しては，デバイスに依存しないレベルで正確に見積ることは困難である．ただし，いくつかの実験を通して，要素数と動的電力には正の相関があること，FPGA 上での各ブロックの動的電力はトグル回数に影響をうけ，そのトグル回数は GPC の段数に関連することが観測された．したがって直接的ではないが，段数，要素数を減らすことが電力削減にも有効であると考えることが可能である．

2.4 圧縮回路合成の基本フロー

GPC を用いた圧縮回路合成に対する既存のヒューリスティックは，基本的に以下のフローに基づいている：

(1) 各ランクのドットを GPC を用いてカバーする．
(2) カバーされた入力ドットを削除し，GPC の出力に相当するドットを加えて，ドットダイアグラムを更新する．
(3) ドットダイアグラムが指定された高さ以下になるまで，上記の処理を繰り返す．

これらのプロセスの中で，ドットをカバーするのに使用した GPC が適切に接続され，圧縮回路が構築される．

この枠組みで実現されるヒューリスティックを特徴付けるポイントとして，以下の要素があげられる．

- 使用する基本構成要素の種類
  ASIC の場合は通常 (3;2) および (2;2) カウンタが用いられる．(7;3) カウンタ等，より大きいカウンタを用いることも可能である[10]．FPGA の場合には，対象としている FPGA を構成する LUT の入力数 k に対応して，入力数 k 以下の GPC 集合が用いられることが多いが，キャリーチェイン構造を利用し，k より多い入力数の GPC を利用することも可能である[11]．ただしこの場合 GPC の面積，遅延コストは GPC の種類によって一定ではなくなる．
- 次の繰り返しに移る条件
  カバーできるドットがなくなってから次に移る手法（以下，手法 1 と呼ぶ），Dadda の手法のように，高さが前もって定めた値まで縮小できたら次に移る手法（以下，手法 2 と呼ぶ）等が 考えられる．
- ドットをカバーする GPC の選択方法
  あらかじめ各 GPC に優先順位を与えておき，それにしたがって選択することが多い．優先順位は最小化したいコストに対応したもので，各 GPC の実現コストが同じであるならば，入力数から出力数への削減率や削減できるドット数が用いられることが多い．

3. ILP による定式化

本稿では，遅延と電力を考慮したマルチオペランド加算器合成問題を，最小段数の下での GPC 数最小化問題ととらえ，ILP 問題として定式化する．主要なアイデアは，前節で示した，繰り返しドットダイアグラムを圧縮していく過程そのものを ILP 問題として実現することである．すなわち，圧縮の各ステージでのドットダイアグラムとそこで使用する GPC の個数を変数で表し，ドットダイアグラムの変化を表現する．

Vol.2011-SLDM-150 No.8
2011/5/19

表 1 基本的な変数
Table 1 Basic variables

変数	意味
$y_{xx}(i,j)$	ステージ i でランク j に対して使用された GPC_{xx} の個数
$x(i,j)$	ステージ i に入った段階での，各ランク j のドット数
$z(i)$	ステージ i でのドットダイアグラムの高さ
st	ステージ数

表 1 に本定式化で用いる主要な変数を示す．$y_{xx}(i,j)$ は，繰り返しのステージ i において，ランク j を最下位ビットとして使用された GPC_{xx} の個数を示す．ここで xx は GPC の種類を表し，入力数を 6 と仮定した場合，集合 $\{06,05,04,03,02,15,14,13,12,23,22\}$ の要素である．例えば，y_{15} は GPC(1,5;3) の個数を表す．

目的関数は次の式の値を最小化することである．

$$\alpha \times st + \sum_{i,j}(y_{06}(i,j) + y_{05}(i,j) + y_{04}(i,j) + y_{03}(i,j) + y_{02}(i,j) + y_{15}(i,j) + y_{14}(i,j) + y_{13}(i,j) + y_{12}(i,j) + y_{23}(i,j) + y_{22}(i,j))$$

ここで，α は少なくとも GPC 総数と比較して，第一目標が段数の最小化になる程度に大きな値であるとする．

各ステージ i の各ランク j のドット数は $x(i,j)$ によって表される．ここで，$0 \leq j \leq W-1$ であり W は最終的なビット幅である．初期ドットダイアグラムを表す $x(0,j)$ 集合は入力として与えられるものとする．

$x(i-1,j)$ と $x(i,j)$ の間には次の関係が存在する．

$$\begin{aligned} x(i,0) = {} & x(i-1,0) - 5\cdot y_{06}(i-1,0) - 4\cdot y_{05}(i-1,0) - 3\cdot y_{04}(i-1,0) \\ & -2\cdot y_{03}(i-1,0) - y_{02}(i-1,0) - 4\cdot y_{15}(i-1,0) - 3\cdot y_{14}(i-1,0) \\ & -2\cdot y_{13}(i-1,0) - y_{12}(i-1,0) - 2\cdot y_{23}(i-1,0) - y_{22}(i-1,0) \end{aligned}$$

$$\begin{aligned} x(i,1) = {} & x(i-1,1) - 5\cdot y_{06}(i-1,1) - 4\cdot y_{05}(i-1,1) - 3\cdot y_{04}(i-1,1) \\ & -2\cdot y_{03}(i-1,1) - y_{02}(i-1,1) - 4\cdot y_{15}(i-1,1) - 3\cdot y_{14}(i-1,1) \\ & -2\cdot y_{13}(i-1,1) - y_{12}(i-1,1) - 2\cdot y_{23}(i-1,1) - y_{22}(i-1,1) \\ & +y_{06}(i-1,0) + y_{05}(i-1,0) + y_{04}(i-1,0) + y_{03}(i-1,0) \\ & +y_{02}(i-1,0) - y_{23}(i-1,0) - y_{22}(i-1,0) \end{aligned}$$

$$\begin{aligned} x(i,j) = {} & x(i-1,j) - 5\cdot y_{06}(i-1,j) - 4\cdot y_{05}(i-1,j) - 3\cdot y_{04}(i-1,j) \\ & -2\cdot y_{03}(i-1,j) - y_{02}(i-1,j) - 4\cdot y_{15}(i-1,j) - 3\cdot y_{14}(i-1,j) \\ & -2\cdot y_{13}(i-1,j) - y_{12}(i-1,j) - 2\cdot y_{23}(i-1,j) - y_{22}(i-1,j) \\ & +y_{06}(i-1,j-1) + y_{05}(i-1,j-1) + y_{04}(i-1,j-1) + y_{03}(i-1,j-1) \\ & +y_{02}(i-1,j-1) - y_{23}(i-1,j-1) - y_{22}(i-1,j-1) \\ & +y_{06}(i-1,j-2) + y_{05}(i-1,j-2) + y_{04}(i-1,j-2) + y_{15}(i-1,j-2) \\ & +y_{14}(i-1,j-2) + y_{13}(i-1,j-2) + y_{12}(i-1,j-2) \\ & +y_{23}(i-1,j-2) + y_{22}(i-1,j-2) \end{aligned}$$

あるステージのドットダイアグラムは，1 つ前のステージにおけるドットダイアグラムに対して，使用した GPC に個数に対応するドット数の増減を行うことによって得られる．各 GPC の使用によって，図 2(b) に示された値だけドット数が増減するので，それらに対応した値が係数としてかかっている．

GPC を使用する際，その時点でのドット数を越えてカバーすることはできないため，以下の制約を満たす必要がある．

$$\begin{aligned} x(i,0) \geq {} & 6\cdot y_{06}(i,0) + 5\cdot y_{05}(i,0) + 4\cdot y_{04}(i,0) + 3\cdot y_{03}(i,0) + 2\cdot y_{02}(i,0) \\ & +5\cdot y_{15}(i,0) + 4\cdot y_{14}(i,0) + 3\cdot y_{13}(i,0) + 2\cdot y_{12}(i,0) \\ & +3\cdot y_{23}(i,0) + 2\cdot y_{22}(i,0) \end{aligned}$$

$$\begin{aligned} x(i,j) \geq {} & 6\cdot y_{06}(i,j) + 5\cdot y_{05}(i,j) + 4\cdot y_{04}(i,j) + 3\cdot y_{03}(i,j) + 2\cdot y_{02}(i,j) \\ & +5\cdot y_{15}(i,j) + 4\cdot y_{14}(i,j) + 3\cdot y_{13}(i,j) + 2\cdot y_{12}(i,j) \\ & +3\cdot y_{23}(i,j) + 2\cdot y_{22}(i,j) \\ & +y_{15}(i,j-1) + y_{14}(i,j-1) + y_{13}(i,j-1) + y_{12}(i,j-1) \\ & +2\cdot y_{23}(i,j-1) + 2\cdot y_{22}(i,j-1) \end{aligned}$$

繰り返しが終了するのはドットダイアグラムの高さが 3 以下になったときである．各ステージ i での高さを表す変数を $z(i)$ とする．$u(i,j)$ を，$x(i,j) = z(i)$ のときに 1 になる変数とすると，すべての i,j に対して，次の式が成り立つ．

$$z(i) \geq x(i,j)$$

$$z(i) \leq x(i,j) - \beta \times (1 - u(i,j))$$

$$\sum_j u(i,j) \geq 1$$

最初の式は $z(i)$ がステージ i の全てのランクのドット数以上であることを意味する．次

の式において，β は十分大きな値に設定することで，$z(i)$ がある列 j の高さに等しいときに $u(i,j)=1$ となる．最後の式により，少なくとも 1 つの列において $x(i,j)$ が $z(i)$ と等しくなり, $z(i)$ がドットダイアグラムの高さであることが保証される．

オペランド数圧縮に必要な段数を最小化することを目的としているため，繰り返し圧縮の過程において，各ステージのドットダイアグラムの高さは単調減少であるとし，最終的に 3 以下になるとする：

$$z(i-1)-z(i)\geq 0$$
$$z(L-1)\leq 3$$

ここで，L は想定する最大ステージ数とする．実際のステージ数を表現するために変数 $w(i)$ を導入する：

$$z(i)-\gamma w(i) <= 3$$
$$st=\sum_i w(i)$$

γ は上記 β と同様に，$z(i)>3$ であるときに $w(i)$ が 1 となるようにするために導入した，十分大きな値である．したがって 2 番目の式はドットダイアグラムの高さが 3 より大きいステージの数を表しており，st を最小化することが段数最小化を意味する．

この問題の解いた結果得られるのは，各ステージにおける各ランクのドット数と，使用する GPC の種類とその個数だけであり，GPC による圧縮回路の構造自体が得られる訳ではないため，得られた情報から圧縮回路を構築する過程が別途必要である．

4. 関 連 研 究

GPC による圧縮回路合成問題を ILP 問題に定式化して解くアプローチは, 9) で提案されている．そこでは，圧縮回路を構成する各 GPC, 及び，GPC 間，GPC と入出力間の接続に対して変数を割り当て，GPC の段数の最小化を目標とする．要素数に関しては，入力数が 5 以上の GPC を使用するという制約を与えることで，間接的に削減している．また問題の複雑度を減少させるために，最終的に得られる圧縮回路の構造にいくつかの制約を与えているため，厳密性は損なわれている．

一方，本手法は使用する GPC の種類・個数を決める処理と圧縮回路を構築する処理を分離し，その前半部分に対して，オペランド数圧縮フローに沿った形で ILP による定式化を行っているところ，及び，段数だけでなく要素数も直接的に最小化を目指しているところが異なる．使用する GPC に対する制限は与えていない．回路構築の処理を分離したことにより，段数，要素数は同じでも，より細かい遅延等の値を考慮にいれた接続方法を別途考案・試行することが可能である．また，各 GPC のコストが一様でなかった場合にも，各 GPC のコストの総和の形で全体のコストが与えられるならば，同じ枠組みで拡張可能であると考えられる．

5. 実 験 結 果

オペランド数圧縮回路合成問題を提案手法に基づいて ILP 問題として定式化し，ILOG 社の CPLEX を用いて実験を行った．入力ドットダイアグラムとして，以下の 4 種類を対象とした．

(1) ビット幅が同じオペランドの多重加算 (madd)
ビット幅は 8, 12, 16, オペランド数は 10, 20, 30 で実行.

(2) 8 ビット乗算，12 ビット乗算の圧縮回路部分 (mult)

(3) A + B * C の圧縮回路部分
オペランドのビット幅は一様で，8 および 12

(4) 任意の形状のドットダイアグラム
ビット幅は 8, 12, 16, 最大高さ 30 としてランダムに各列の高さを設定.

比較対象として，以下の 2 つのヒューリスティック（2.3 節参照），

- 手法 1:ドットダイアグラム圧縮のフローにおいて，カバーできるドットがなるまでカバーする手法[5)]
- 手法 2 前もって決められた高さ以下になるまでカバーする手法[7)]

の結果も示す.

表 2 に，手法 1，2，及び，提案手法によって生成された圧縮回路における GPC の最大段数 (lv) と GPC 総数 (gpc) を示す．提案手法に関しては，GPC 数の手法 1，2，に対する比と，ILP 問題を解くのに要した時間（cpu, 単位は秒）も示している（Intel Core2 Quad Q9550, 2.83GHz 上で測定）．表から分かるように，手法 1 と 2 で生成された圧縮回路に対して，それぞれ 11-27%, 2-12% 要素数が少なくなっており，ヒューリスティックにはさらなる改良の余地が残されている可能性が確認された．段数に関しては概ね最小解が得られていたが，ランダムにドットダイアグラムを生成した場合などで，最小解が得られていない場合も見られた．完全にランダムな形状のドットダイアグラムが現実的な例として現れる可能性は少ないかもしれないが，異なる幅のオペランドや定数との乗算が含まれるような演算では madd の場合のような一様な形状にはならないため，このような場合に対するヒューリ

表 2 実験結果
Table 2 Expreimental results

			手法 1		手法 2		提案手法				
type	bit	op	lv	gpc	lv	gpc	lv	gpc	/手法 1	/手法 2	cpu
madd	8	10	2	23	2	20	2	19	0.83	0.95	0.34
madd	8	20	3	51	3	48	3	43	0.84	0.90	0.72
madd	8	30	4	78	4	75	4	69	0.88	0.92	8.24
madd	12	10	2	35	2	30	2	29	0.83	0.97	1.5
madd	12	20	3	77	3	72	3	66	0.86	0.92	3.94
madd	12	30	4	118	4	111	4	105	0.89	0.95	7.86
madd	16	10	2	47	2	40	2	39	0.83	0.98	63.97
madd	16	20	3	103	3	96	3	88	0.85	0.92	7.88
madd	16	30	4	158	4	147	4	141	0.89	0.96	109.22
mult	8		2	15	2	12	2	11	0.73	0.92	0.14
mult	12		2	37	2	34	2	31	0.84	0.91	85.56
mac	8		2	17	2	14	2	13	0.76	0.93	0.16
mac	12		2	40	2	38	2	34	0.85	0.89	4.33
rand	8	30	4	57	4	56	3	50	0.88	0.89	0.34
rand	12	30	4	67	4	64	3	57	0.85	0.89	1.11
rand	16	30	4	100	4	93	3	82	0.82	0.88	1.92

スティックの強化の必要があると思われる．

一方，提案手法の実行時間については，表 2 のデータに対しては最大 100 秒程度であったが，これより大きなデータ，例えば 16 ビットの乗算器に対しては，30 分程度でメモリーオーバー (2GB) となり結果が得られなかった．変数の数などを厳密性を損なわずに減らせる可能性はあるが，16 ビット madd の結果に見られるように，オペランド数が少ないのに時間がよりかかってしまうケースもあり，さらなる解析が必要と思われる．

6. おわりに

本稿では，マルチオペランド加算器を構成するオペランド数圧縮回路の合成過程を，使用する GPC の選定とそれを用いた回路の構築の 2 つのフェイズに分解し，前半部分を段数および GPC 数の最小化を目的とした ILP 問題として定式化して解くアプローチを提案した．実験結果より，最小解は既存のヒューリスティックから得られ値より少なくとも数%，最大で 27%小さい値になることや，ドットダイアグラムの形状が不均一である場合には，段数も最小値が得られない場合があることが判明した．現状では本手法を 16 ビット以上の乗算器には適用できず，スケーラビリティの問題があるが，実験で得られた最小解はヒューリスティックの性能の評価やその改良に対する指針を与えるものとして意味があると考えられる．

謝辞 本研究において貴重なご助言をいただいた埼玉大学堀山貴史准教授に感謝いたします．本研究は一部，JST CREST ULP プロジェクトの支援による．

参 考 文 献

1) C.Wallace, “A suggestion for a fast multiplier,” IEE Transactions on Electronic Computers, vol.EC-13, pp.14–17, 1964.
2) L.Dadda, “Some schemes for parallel multipliers,” Alta Frequenza, vol.34, pp.349–356, 1965.
3) V.G. Oklobdzija and D.Villeger, “Improving multiplier design by using improved column compression tree and optimized final adder in cmos technology,” IEEE Transactions on VLSI Systems, vol.3, no.2, 1995.
4) P.Stelling, C.Martel, V.G. Oklobdzija, and R.Ravi, “Optimal circuits for parallel multipliers,” IEEE Transaction on Computers, vol.47, no.3, pp.273–285, 1998.
5) H.Parandeh-Afshar, P.Brisk and P.Ienne., “Efficient synthesis of compressor trees on FPGAs,” in *Proc. Asia and South Pacific Design Automation Conference*, pp.138–143, January 2008.
6) H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Exploiting fast carry-chains of fpgas for designing compressor trees,” the 19th International Conference on Field-Programmable Logic and Application, pp.242–249, August2009.
7) T.Matsunaga, S.Kimura, and Y.Matsunaga, “Multi-operand adder synthesis on fpgas using generalized parallel counters,” Proc. Asia and South Pacific Design Automation Conference, pp.337–342, January2010.
8) 松永多苗子, 木村晋二, 松永裕介：キャリーチェインを用いたマルチオペランド加算器の FPGA 向け低電力合成手法, 情報処理学会研究報告, Vol.2011-SLDM-148, No. 1, pp.1-6, 2011 年 1 月.
9) H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Improving synthesis of compressor trees on FPGAs via integer linear programming,” in *Proc. Design, Automation and Test in Europe*, pp.1256–1261, March 2008.
10) A.K. Verma and P.Ienne, “Automatic synthesis of compressor trees: Reevaluating large counters,” Design, Automation, and Test in Europe, pp. 443–448, 2007.
11) H.Parandeh-Afshar, A.Neogy, P.Brisk, and P.Ienne, “Improved Synthesis of Compressor Trees on FPGAs by a Hybrid and Systematic Design Approach,” International Workshop on Logic and Synthesis, pp.193–200, June2010.