

並列プレフィックス加算器合成問題への 乱択アルゴリズムの適用について

松永 多苗子^{1,a)} 松永 裕介²

概要：並列プレフィックス加算器は、基本となる演算を実行する順序の決め方に大きな自由度がある。そのため、様々な特徴をもったアーキテクチャが提案されているとともに、与えられた制約や要求に合う回路構造を自動的に決定するアルゴリズムも研究されている。本稿では並列プレフィックス加算器合成問題とそれに対する既存の手法をサーベイするとともに、モンテカルロ木探索を用いたアプローチをもとに確率的要素を取り入れたアプローチの可能性について考察するものである。

キーワード：並列プレフィックス加算器，モンテカルロ木探索，シミュレーテッド・アニーリング

On Randomized Approach for Parallel Prefix Adders Synthesis

MATSUNAGA TAEKO^{1,a)} MATSUNAGA YUSUKE²

Abstract: Parallel prefix adders have a high degree of freedom for the order to execute each basic operation. Various architectures are known, and also several approaches have been proposed for automatic generation of parallel prefix adders under various constraints and objective functions. This paper surveys the formalization of this problem and the existing approaches, then discusses the possibility of another approach with randomness, such as Monte-Carlo tree search.

Keywords: parallel prefix adder, Monte-Carlo tree search, simulated annealing

1. はじめに

算術演算回路は様々な応用分野において、システム全体の品質に影響を与える重要な回路要素である。特に2入力加算は最も基本的な演算であり、乗算等の他の演算の構成要素としても用いられる使用頻度の高い演算であるため、従来より数多くの加算器アーキテクチャが提案されている[1]。中でも並列プレフィックス加算器は、先見桁上げの概念を一般化した手法により桁上げ伝搬を高速化するとともに[2]、構造をあらかじめ固定するのではなく制約に応じて自動的に決定するアルゴリズムも提案されてい

る[3], [4], [5], [6]。

並列プレフィックス加算器のアーキテクチャは、プレフィックス演算の実行順序を表現したグラフにより特徴付けられることが多い。そのノード数（実行する演算の個数）や段数（逐次的に実行する演算数）を、回路の面積や遅延時間のラフな見積もりとして扱うことにより、様々なアルゴリズムが提案されている。また、シミュレーテッド・アニーリングやモンテカルロ木探索法など、確率的要素を取り入れたアプローチも現れている[8], [9]。

本稿では並列プレフィックス加算器合成問題に対する既存の手法を総括するとともに、確率的な要素をもつ手法の可能性について議論を行うものである。以下、2節において並列プレフィックス加算器合成問題を定義し、3節において既存の手法を概観する。4節では、確率的な要素を持つ手法の一つとしてモンテカルロ木探索を用いた手法につ

¹ 日本文理大学工学部情報メディア学科
〒870-0397 大分市一木 1727

² 九州大学大学院システム情報科学研究院

^{a)} matsunagatk@nbu.ac.jp

いて説明し、実験を交えながら考察を行う。

2. 並列プレフィックス加算器

N 個の入力 $x_N, x_{N-1}, \dots, x_1$ と結合則を満たす任意の演算 $\circ$ が与えられたとき, N 個の出力 y_i を $y_i = x_i \circ x_{i-1} \circ \dots \circ x_1$ によって計算するものをプレフィックス計算と呼ぶ。これは, i 番目の出力 y_i は $j \leq i$ となる入力 x_j のみに依存することを意味する。

n ビットの 2 進加算の入力を $A = a_n, \dots, a_1, B = b_n, \dots, b_1$, 出力を和 $S = s_n, \dots, s_1$, 及び, キャリー出力 c_n とすると, 各ビットの和 s_i とキャリー出力 c_i は, $s_i = a_i \oplus b_i \oplus c_{i-1}$ と $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$ で計算される。この n ビット加算は, 1 ビット信号に対する桁上げ生成/伝搬関数 (g, p) と, それを連続する複数ビットのグループに対する演算に拡張した桁上げ生成/伝搬関数 (G, P) を用いると以下のように計算できる。

- g, p の生成: $g_i = a_i \cdot b_i$, $p_i = a_i \oplus b_i$
- プレフィックス処理: G, P を用いて $c_i = G_{[i:1]}$ を計算

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]} & \text{otherwise} \end{cases}$$

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j \\ P_{[i:k]} \cdot P_{[k-1:j]} & \text{otherwise} \end{cases}$$

- s_i の生成: $c_i = G_{[i:0]}$, $s_i = p_i \oplus c_{i-1}$

このうちプレフィックス処理の部分は, 演算 $\circ$ を以下のように定義することによってプレフィックス計算とみなすことができる。

$$\begin{aligned} (G, P)_{[i:j]} &= (G, P)_{[i:k]} \circ (G, P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, \\ &\quad P_{[i:k]} \cdot P_{[k-1:j]}) \end{aligned}$$

プレフィックス計算における演算順序は, グラフの形で表現することができる。 N 入力プレフィックスグラフは, N 個の入力に対するプレフィックス計算における各演算 $\circ$ をノードとした非巡回有向グラフ (directed acyclic graph: DAG) であり, 各出力の値を計算するための演算の実行順序を表現したものである。ノードの 2 つの入力は 2 項演算 $\circ$ の 2 つのオペランドに対応し, ノード v_1 が v_2 のオペランドであるとき, v_1 から v_2 へのエッジが存在する。 v_1 を v_2 のファンインノード, v_2 を v_1 のファンアウトノードと呼ぶ。

並列プレフィックス加算器の面積や遅延時間等の特性は, プレフィックス処理部の構成に依存し, その構成はプレフィックスグラフによって表現される。並列プレフィックス加算器は多くの場合, プレフィックスグラフのノード数, 入力から出力への段数, 1 つのノードから出力される

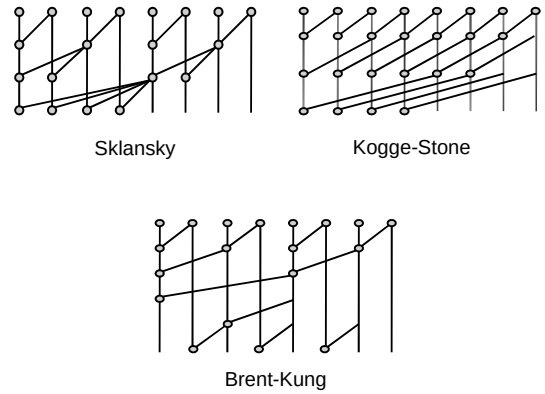


図 1 並列プレフィックス加算器の例

エッジの本数 (ファンアウト数) を用いて特徴づけられる。図 1 は特徴的な 3 つの並列プレフィックス加算器を示している。

3. 既存手法

並列プレフィックス加算器合成問題は, プレフィックスグラフをベースにして, そのノード数や段数を最小化する問題として扱われることが多い。各ビットごとに入力到達時刻や出力要求時刻が異なる, といった個別状況を考慮するアプローチもある。また, 電力消費を考慮するために, 最大ファンアウト数に制約を与えたり, スイッチング確率を考慮する場合もある。

プレフィックスグラフの i 番目のビットに対する出力を計算するプレフィックスグラフの構造は, i のカタラン数通りある [6]。ただし, カタラン数は以下で計算される。

$$catalan(i) = \frac{1}{i+1} \binom{2i}{i}$$

ビット幅 n に対して可能なプレフィックスグラフ集合 G_n のサイズ $|G_n|$ は, $|G_n| = catalan(n-1) * catalan(n-2) * \dots * catalan(0)$ となり, ビット幅 8 としたときでも 332972640 となり, 厳密解を求めるのは事実上困難である。既存の手法では, 何らかの制約を加えて探索空間を狭めたり, ヒューリスティックを用いて問題に対応している。例えば, 局所変換によるヒューリスティック [3], 動的計画法の適用 [4], [5], 制約や効果的な枝刈りによる解空間の縮小化 [5], [6] などがある。プレフィックスグラフ上で定式化された問題に対するアプローチは, 解にある程度の差はあるにせよ, 実用的な時間内で, ノード数・段数の観点からは最適に近い結果を出すことが概ね可能になっていると言える。この場合, 残された課題は, 評価指標をより正確にするための枠組みで, 特に, プレフィックスグラフ上では評価しにくい消費電力を考慮することがその一つに挙げられる。

一方で, シミュレーテッド・アニーリングによる定式化 [9] や, モンテカルロ木探索の適用 [8] といったアプローチも提案されている。これらの手法は, 上で述べた手法よ

り実行時間はかかるが、逆に言えば、時間をかければ解が良くなる可能性がある。また、例えばテクノロジマッピングや配置配線後の評価値といった、より正確な指標を導入するための枠組みとして用いることも可能性として考えられる。

そこで、以下では [8] で提案したモンテカルロ木探索に基づく加算器合成手法とその改良について述べ、いくつかの実験をもとに評価・考察を行う。

4. モンテカルロ木探索に基づくアプローチ

4.1 モンテカルロ木探索

モンテカルロ木探索とは、ランダムサンプリングを用いて期待値等を計算するモンテカルロ法と、木の探索アルゴリズムを組み合わせた手法である [7]。図 2 に探索における 1 サイクルの動作とトップレベルアルゴリズムを示す。

図 2 において各節点は部分解に相当し、矩形の節点の一つの解に相当する。ここで V_0 は探索の開始点（探索木の根のノード）を示す。一回の処理は、展開されている末端ノードを求め (TREE_POLICY)、そのノードに関する評価値を計算し (DEFAULT_POLICY)、その評価値を反映させる (BACKUP) 処理から構成される。この処理を与えられた計算時間や繰り返し回数制限の範囲内で繰り返し、最後に最も評価値の高かったノード返す。

展開する子ノード V_j の選択には、以下の UCT (Upper Confidence Bound fo Trees) で示される指標を用いる。

$$UCT = \bar{x}_j + 2C_p \sqrt{\frac{2 \ln n}{n_j}} \quad (1)$$

ここで、 n は他の選択肢も含めた総試行数、 n_j は V_j における試行数であり、 $\bar{x}_j$ はその時点での V_j の期待値（試行における評価値の平均）である。2 項目の $2C_p$ は、期待値が $[0,1]$ の範囲と異なる場合の補正のための係数である。

4.2 モンテカルロ木探索を用いた並列プレフィックス加算器合成

モンテカルロ木探索を並列プレフィックス加算器合成に適用するにあたって、1 つの解（部分解を含む）であるプレフィックス構造を表すために、図 3 に示すような、入力範囲の幅で整列されたプレフィックスグラフを考える。

プレフィックス演算に対応する各ノードは、一連のビット範囲に対する G, P 演算を実現する。 r 行 c 列のノードの場合、幅が $r+1$ で最上位ビットが c である範囲の G, P を計算している。最終的にすべての出力 (i 行 i 列のノード) に対して 0 から i の範囲の計算を実行している状態が一つの解となる。

この例は 4 ビットのプレフィックス構造の一つを表している。 i 行 j 列のノード $v_{i,j}$ ($i \leq j$) は、 $i+1$ 個の入力に対するプレフィックス計算の中間結果を表わしている。演算

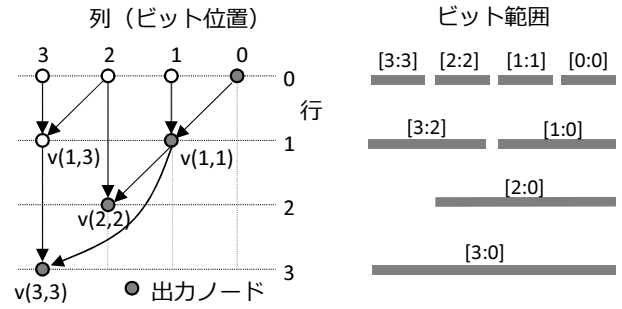


図 3 ノードを入力範囲の幅で整列したプレフィックスグラフと、各ノードに対応するビット範囲

の 2 つのオペランドは、連続した入力範囲に対する中間結果となっている。

例えば、図 3 の $v_{3,3}$ は 2 つのファンイン $v_{1,3}, v_{1,1}$ を持ち、 $v_{3,3} = v_{1,3} \circ v_{1,1}$ という演算結果を表している。 $v_{1,3}$ は入力 2 から 3 に対する演算結果、 $v_{1,1}$ は入力 0 から 1 に対する演算結果であり、それらに演算を施した結果 $v_{3,3}$ は入力 0 から 3 に対する演算結果を表わしている。

モンテカルロ木探索を適用する方針は以下の通りである [8]。

- 初期状態：レベル 0 の n 個のノード ($v(0,i), 0 \leq i \leq n-1$) からなるプレフィックスグラフ。
- 毎回の処理：プレフィックスグラフ中の 2 つのノードに対して演算を行い 1 つのノードを生成。
- 終了判定：すべての出力 ($v(i,i), 0 \leq i \leq n-1$) がグラフに含まれること。

探索における毎回の処理で 2 つのノードを組み合わせるといことは、 $[lsb1 : msb1]$ の範囲の演算と $[lsb2 : msb2]$ ($lsb1 \leq lsb2, msb1 \leq msb2$ とする) の範囲の演算を結合して $[lsb2 : msb2]$ の範囲の演算結果を求めることに相当する。演算は隙間があいてはいけませんが、重なりがあっても構わない。ただし、問題を簡単にするため、現状では重ならない連続した範囲 ($lsb2 = msb1 + 1$) として扱っている。

最終状態においては、すべてのビット i において、0 から i までの連続した範囲の演算結果が生成されている（つまりプレフィックスグラフの出力ノードが生成されている）。

1 回の試行では、図 2 に示すように TREE_POLICY、DEFAULT_POLICY、BACKUP を繰り返す。TREE_POLICY では子ノードが展開済みかどうかを判断し、展開済の場合はより深く探索するノードを選択、展開済み出ない場合は次に展開するノードを選択する。

DEFAULT_POLICY では、ランダムサンプリングを行ってプレフィックスグラフの 1 つの解を作成・評価する。BACKUP でその評価値をルートノードに向かって伝搬し、指標を更新する。

[8] では DEFAULT_POLICY において単純なランダムサンプリングを行っており非常に効率が悪かった。それに

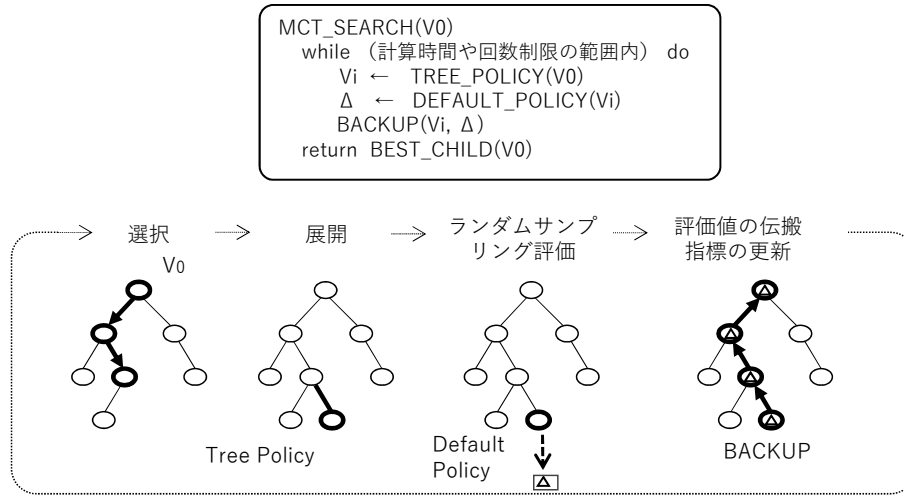


図 2 モンテカルロ木探索における処理の流れ

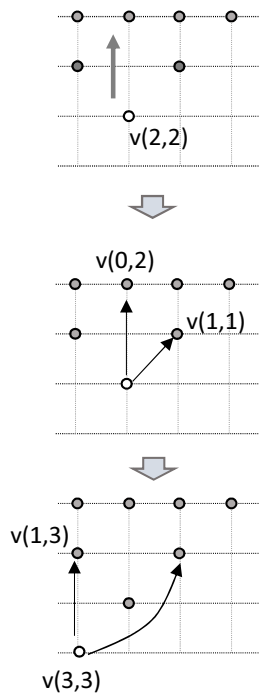


図 4 DEFAULT_POLICY における解の生成

対して、今回の実装では DEFAULT_POLICY が呼ばれた段階で生成されていない出力ノードをターゲットとして、すでに部分解に含まれているノードを使って計算する方法をとっている。図 4 にその流れを示す。この部分解において出力ノード $v(2,2)$ が生成されていないかとする。そして、その列で生成されているノード（この場合 $v(0,2)$ ）を一つのファンインとし、もう片方のファンイン $v(1,1)$ を用いて $v(2,2)$ を生成する。この処理をビット位置の昇順に行うことにより、必ず部分解に含まれているノードのみから出力ノードを生成することになる。

4.3 考察

図 5 に、ビット幅 8 の加算器に対して、段数制約をそれぞれ 3（最小段数）と 7（最大段数）に設定した場合の、各試行で得られた最適解の段数・ノード数の値の変化をグラフで示す。どちらの場合も、最終的に段数制約を満たす値に落ち着いているが、段数制約 7（すなわち、リップルキャリー型）の方が、制約を満たす値への収束が遅い。現行のコスト関数においては、段数制約を超えた場合にのみペナルティを与えているので、制約がない状態だとノード間のコストの差がつきにくいと考えられるが、それだけでなく DEFAULT_POLICY での処理の影響がないかどうか、確認する必要がある。

5. おわりに

本稿では、並列プレフィックス加算器合成問題と既存のアプローチについての概観を述べるとともに、確率的要素をもったアプローチの試行結果を示した。モンテカルロ木探索法を並列プレフィックス加算器合成問題へ適用可能であることは確認できた。ただし、既存のアプローチの方がより高速に同様の結果が得られるため、プレフィックスグラフのノード・段数をコストとして用いる限りにおいては、モンテカルロ木探索を適用する必要性はないと思われる。今後の可能性については、より複雑な指標の評価が組み込まれる枠組みとできるかどうか、さらなる検討が必要である。また、その前提として、実装上の改良も必要である。

複雑な指標を組み込んだり、時間をかければ解が改善できる仕組み、という意味では、シミュレーテッド・アニーリングなど、他にも可能性はある。今後他の手法との比較も行っていく予定である。

参考文献

- [1] I. Koren, *Computer Arithmetic Algorithms*. A K Peters, Ltd., 2002.

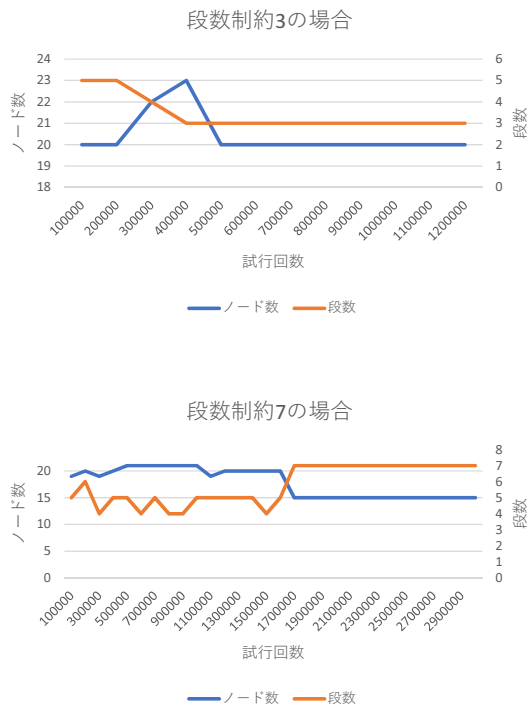


図 5 8 ビット加算器

- [2] N. H. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*, chapter 10. Datapath Subsystems, pages 637–711. Addison Wesley, 2004.
- [3] R. Zimmermann, “Non-heuristic optimization and synthesis of parallel-prefix adders”, In *International Workshop on Logic and Architecture Synthesis*, pages 123–132, December 1996.
- [4] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, “An algorithmic approach for generic parallel adders”, In *ICCAD*, pages 734–730, November 2003.
- [5] Taeko Matsunaga and Yusuke Matsunaga, “Timing-Constrained Area Minimization Algorithm for Parallel Prefix Adders”, in *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E90-A, no.12, pp.2770-2777, December 2007.
- [6] S. Roy, M. Choudhury, R.Puri and David Z. Pan, “Towards Optimal Performance-Area Trade-Off in Adders by Synthesis of Parallel Prefix Structures”, in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.33, no.10, pp.1517-1530, October 2014.
- [7] C. Browne, E. Powley, D. Whitehouse, S. Lucas, “P.I.Cowling, P.Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton”, “A Survey of Monte Carlo Tree Search Methods”, in *IEEE Transactions on Computational Intelligence and AI in Games*, vol.4, no.1, pp.1-49, March 2012.
- [8] 松永多苗子, 松永裕介, “モンテカルロ木探索を用いた並列プレフィックス加算器合成手法”, DA シンポジウム, p.140-144, 2016.
- [9] 本 敬之, 金子峰雄, “シミュレーテッド・アニーリングを利用した並列プレフィックス加算器の構成”, 信学技法 VLD2016-127, pp.139-144, 2017.