計算機アーキテクチャ 115－3
設 計 自 動 化 78－3
（1995. 12. 14）

割り当て確率に基づくデータフローグラフのスケジューリング手法

松浦 昭洋　野村 亮　名古屋 彰

e-mail: matsu@cslab.kecl.ntt.jp
NTT コミュニケーション科学研究所
〒619-02 京都府相楽郡精華町光台 2

あらまし　高位合成 (ハイレベルシンセシス) では, 動作記述のコンパイル時に得られるコントロール / データフローグラフに対する, (1) リソース最小化 (2) 処理ステップ数最小化 , を実現する実用時間でのスケジューリングが重要になる. 本稿では, 時間制約 (処理ステップ数一定) の条件下で, (1) の中で特に演算コストを最小化するための, 新たな多項式時間スケジューリング手法を提案し, その有効性を示す. その中ではまず, 各ステップで必要とされる演算装置数の期待値を従来より正確に反映した, 確率ベースの演算コストを新たに定義し, 続いて, その演算コストと minimax 原理による選択法を用いたスケジューリング手法を示す. 合わせて, いくつかのベンチマークに本手法を適用した結果も示す.

Exact-Probability Oriented Scheduling of Data-Flow Graph

Akihiro Matsuura, Ryo Nomura, Akira Nagoya

NTT Communication Science Laboratories
2 Hikaridai, Soraku-gun, Seika-cho, Kyoto, 619-02 Japan

Abstract　In High Level Synthesis, it is important to schedule operators in a control/data-flow graph within a practical time which results in (1)resource minimization and (2)execution time minimization. In this paper, we propose a new time-constrained scheduling algorithm with a polynomial time complexity for the operator cost minimization. First, we define the operator cost which accurately represents the expected number of operators in each control step. Next, we present our scheduling algorithm which makes the best use of the proposed operator cost and the decision of the minimax principle. Finally, we show some experimental results.

1 はじめに

ハイレベルシンセシス (高位合成) において, 動作記述のコンパイル時に得られる中間的表現であるコントロール / データグラフのスケジューリングは, 最終的に必要となるリソース数やチップ面積, 実行ステップ数等に直接反映する重要なフェーズである. そのためのアプローチは, 大きく分けると次の 2 通りである:

1. 実行ステップ制約の下での, リソース最小化
2. リソース制約の下での, 実行ステップ数最小化

今回考える 1 のアプローチでは, 従来, 大域的な最適解を与える手法として 0-1 整数計画法が知られている [1]. しかし, この手法では, ステップ数, 演算数に関して指数オーダの計算量が必要であるため, 実用的な観点から, これまでにもヒューリスティックな解法が提案されてきた [2, 3, 8]. この中で FDS[3] は, 比較的計算量が軽く, 良いスケジューリングが可能なため, しばしば用いられているが [4, 5, 6], そこで採用されている演算コストの基になる割り当て確率は, 正確な確率を求める困難さもあって, 近似の粗いものであった.

そこで本稿では, まず, これまでの演算コストに代わる, より近似の精度の高い演算コストを提案する. これは, 各ステップで必要とされる演算数の期待値を従来より強く反映させることの出来る, 確率ベースの演算コストである. 続いて, この演算コストの, 正確な確率に対する近似の良さを生かす, minimax 原理による選択法を用いたスケジューリング手法を提案する. さらに, ベンチマークのデータフローグラフに対する本手法の適用結果についても述べる.

2 問題の設定

まず, 問題を定式化するために用語を準備する.

Definition G : コントロール / データフローグラフ
$Op = \{O_i \mid 1 \leq i \leq l\}$: G における演算全体
$S = \{S_j \mid 1 \leq j \leq m\}$: コントロールステップ全体
$Type = \{t_k \mid 1 \leq k \leq n\}$: 異なるタイプの演算全体
S_{E_i} : O_i を割り当て得る最も早いコントロールステップ
S_{L_i} : O_i を割り当て得る最も遅いコントロールステップ
$T_i = \{S_j \mid E_i \leq j \leq Li\}$: 演算 O_i のタイムフレーム
$w_{t_k}(>0) : t_k \in T$ に対するウェイト

この時, 演算コストを次のように定式化する:

$$Operator\ Cost = \sum_{k=1}^{n} w_{t_k} * max_{j \in [1,m]} N_{j,t_k}$$

但し, N_{j,t_k} は S_j における t_k タイプの演算の実行数.

ここで考えるスケジューリング問題とは, 上のコスト関数を最小化するスケジュールとコストを見つけるというものである. これはＮＰ完全問題の一つであるので, 我々はここで, 右辺の N_{j,t_k} を的確に表現するためのヒューリスティックな演算コストを新たに定義し, 近似解法によって解くアプローチを取る.

3 確率ベース演算コスト

3.1 従来手法

従来, コントロール / データフローグラフのスケジューリングにおいて, 演算の各ステップへの割り当て確率として用いられてきたものは, 元々 Nagle[7] の attraction algorithm の中で用いられた確率付与の考え方を, 演算コスト最小化のために適用したものであった.

そこでは, 演算 O_i をタイムフレーム $T_i = \{S_j \mid E_i \leq j \leq Li\}$ 中の S_j に割り当てる確率は,

$$Prob(O_i, S_j) = \frac{1}{L_i - E_i + 1}$$

ステップ S_j の演算コストは,

$$Operator\ Cost_j = \sum_{i=1}^{n} Prob(O_i, S_j)$$

によって与えられる.

以下ではまず, FDS 以来しばしば例題として利用されているデータフローグラフ (図 1) に対して従来からの手法を適用して, 次節で, その問題点を明確にする.

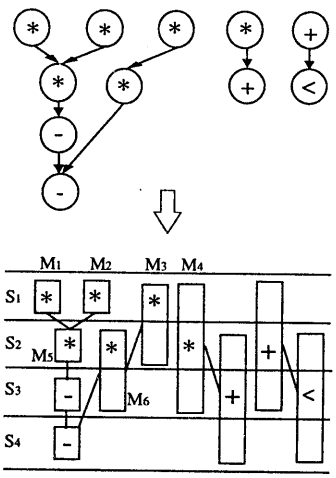

図 1: データフローグラフとタイムフレームグラフ

以下, 簡単のため乗算 ($M_1 \sim M_6$) に絞って考えていくことにする. またしばらくは, 乗算は一ステップで実行される, つまりシングルサイクル演算と仮定しておく.

前述の割り当て確率と演算コストを, 例えばステップ S_1 に対して求めると,

$$\begin{aligned} Operator\ Cost_1 &= \sum_{i=1}^{4} Prob(M_i, S_1) \\ &= 1 + 1 + \frac{1}{2} + \frac{1}{3} = \frac{17}{6} \end{aligned}$$

である.(図 2)

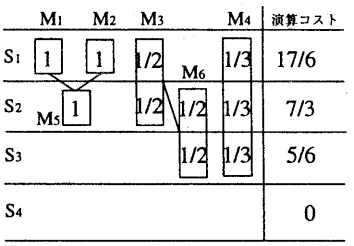

図 2: M_1～M_6 に対する従来の割り当て確率とコスト

3.2 従来手法の問題点

全てのスケジュールが等しく起こり得る，という仮定の下では，M_1 や M_4 に対する確率は，割り当て確率として適当と言える．しかし，M_3 と M_6 については，演算(ノード)の依存関係を考えると，実際には以下の3通りの組合せしかない.(図3)

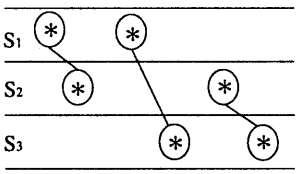

図 3: M_3 と M_6 の割り当て可能性

したがって，各ステップへ割り当てる確率としてより適切なのは，

$$Prob'(M_3, S_1) = \frac{2}{3}, \quad Prob'(M_3, S_2) = \frac{1}{3}$$

$$Prob'(M_6, S_2) = \frac{1}{3}, \quad Prob'(M_6, S_3) = \frac{2}{3}$$

であり，M_3 と M_6 に関する各ステップの演算コストは，S_1, S_2, S_3 とも $\frac{2}{3}$ である.(図4) 上記のように，従来法はノードの依存関係を考慮していないため，割り当て確率としての見積りが大変粗いものであった．

3.3 提案する演算コスト

3.3.1 基本形

前節で求めた $Prob'$ のように，正確な割り当て確率と呼ぶに相応しいものを用いて，演算装置数の正確な期待値として演算コストを表すことが望ましい．しかし，一般のデータフローグラフに対しては，演算数の指数オーダで計算量が増加するため，これまで，そのような確率をベースにしたものは利用されてこなかった．

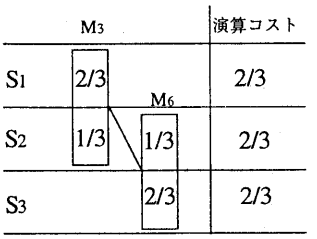

図 4: M_3 と M_6 の割り当て確率 $Prob'$ と演算コスト

しかし，前節の M_3 と M_6 の例において，ステップ毎の演算コストを求めるという点に絞れば，各演算のタイムフレーム内の各ステップにおける $Prob'$ の値を求めることは必ずしも必要ではない．つまり，M_3 と M_6 に関する正確な確率 $Prob'$ の，S_1～S_3 の演算コストに対する寄与 $(=\frac{2}{3})$ は，次の定理から直ちに導くことができる．

Theorem $n \leq m$ とする．n 個の連続的に連なる (依存関係にある) 同種演算を m ステップで割り当てる時，任意のステップ S_j における演算コストは，

$$\begin{aligned} Operator\ Cost_j &= \sum_{\{i \mid S_j \in T_i\}} Prob'(O_i, S_j) \\ &= \frac{n}{m} \quad (\text{定数}) \end{aligned}$$

で与えられる．(図5)

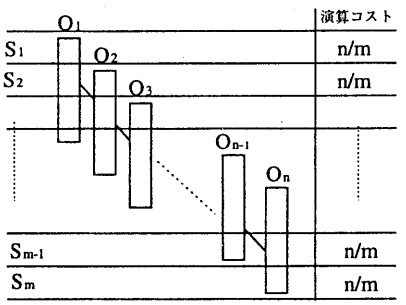

図 5: n 個の依存関係にある演算を m ステップで割り当てた場合の各ステップの演算コスト

この定理によって，依存関係のある同種演算の並びに対しては，全ステップの正確な演算コストを一度の除算で求めることができる．

また，パイプライン演算に関する演算コストを求める場合は，演算の先頭のステップだけを考えることにより，上記と同様の定理が成り立つ．パイプライン化していないマルチサイクル演算に対しては，先頭以外のステップもコストとして加算することによって定理は拡張される．

また, 1 ステップに 2 つ以上の演算を行うチェイン化に対しては, 1 ステップをさらにサブステップに分割して上記定理を適用し, それらをステップ毎に加えれば良い.

3.3.2 一般の場合

グラフが連続する同種のノードのみで構成される場合は, 正確な演算コストが一度の除算によって求まることを前節で述べた. さらに一般のグラフに対しては, 分岐, 併合するノードを考慮する必要があるため, 依然, ノードの数に関して指数オーダで計算が爆発してしまう. そこで, 前節における正確なコストを反映させて, なおかつ計算の軽い, 次のようなヒューリスティックスを導入する:

1. あらかじめデータフローグラフが持っているノードの依存関係を表すデータを元に, 連続して連なるタイムフレーム幅の等しい同種のノードの並び毎に, 分岐, 合併ノードを端点として, クラスタリングする.(図 6)
2. 各クラスターに対して, 前節の正確な確率の和を演算コストとして与える.
3. クラスター全体で, 各ステップ毎に, 2 で求まった演算コストの和をとる.

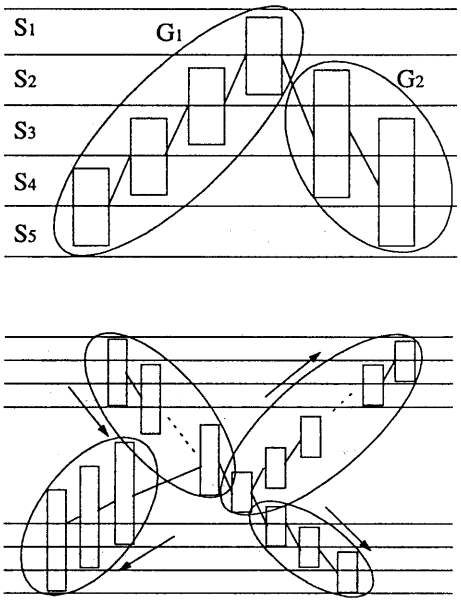

図 6: 連続する演算の並びによるクラスタリング

図 6 の最初のグラフに対する本演算コストの計算例を示す. まず, 左方のクラスター G_1 は, 演算数 4, ステップ数 5 であるので, 前頁の演算コストに関する定理から, ステップ $S_1 \sim S_5$ に割り当てられる確率の和は, 各々 $\frac{4}{5}$. 一方, 右方のクラスター G_2 は, 演算数 2, ステップ数 4 であるから, ステップ $S_2 \sim S_5$ に割り当てられる確率の和は, 各々 $\frac{2}{4} = \frac{1}{2}$. したがって, 各ステップの演算コストは, ステップ毎に各クラスターに対するコストの和をとれば, S_1 の演算コストは $\frac{4}{5}$, $S_2 \sim S_5$ の演算コストは $\frac{4}{5} + \frac{1}{2} = \frac{13}{10}$ である.

この例に関して, 正確な確率によるコスト, 従来の演算コスト, および本手法を適用した場合のコスト比較表を図 7 に記す. 但し, ステップ毎に値を通分してある.

	正確な演算コスト	従来手法	提案する手法
S_1	80/90	45/90	**72/90**
S_2	110/90	75/90	**117/90**
S_3	350/270	450/270	**351/270**
S_4	350/270	450/270	**351/270**
S_5	350/270	225/270	**351/270**

図 7: 図 6 の最初の例に対する正確な確率による演算コストと, 従来法, 本手法による演算コストの比較

これは提案するコストの近似の精度の高さを示す一例である. その他の多くのグラフにおいても, 精度の高さは確認されている.

3.4 計算量について

一般に, コントロールデータフローグラフのスケジューリングを考える時, ノードの依存関係や, 各演算のタイムフレームなどの情報は不可欠なものであり, それらの情報は, コントロールデータフローグラフ生成とスケジューリングの初期の段階で, 各演算ノードに対して付与されるものと考えて良い. 上記演算コストの導出において, 初期のクラスタリングに関しては, ノードの依存関係を利用してタイムフレームの情報を付与していく際に同じタイムフレーム幅のノードをクラスタリングしていけば良いため, 計算量へのオーダレベルでの影響は避けることができる. また, 一旦クラスターが出来上がったグラフで, 本手法による演算コスト導出のためにかかる計算量については, 演算数を n 個, ステップ数を m とすると $O(mn)$ であることが, 次のようにして分かる:

- 各ステップの演算コストの計算に関しては, クラスターの数は高々 n 個であり, 各クラスターには前節の定理による一つの分数をコストとして付与すれば良く, 最悪の場合でも, クラスターの数が演算数と一致して, 従来法と全く同じ計算をすることになる. したがって, 全ステップの演算コストを導くためのの計算量は $m \cdot O(n) = O(mn)$ である.

なお, n 個の連続的に連なる同種演算を m ステップで割り当てるという定理1の条件下での, 演算コスト導出のための計算量は $O(n)$ である. このことは, そのような演算の並びが局所的に現れる一般のデータフローグラフにおいて, クラスター毎に処理することが, 正確な確率を使ったコストに対する近似度が増すのに加えて, 処理時間短縮の効果もあることの, 理論的な裏付けの一つと考えられる.

4 minimax 原理によるスケジューリング

4.1 準備

FDS においては, 各演算の各ステップに均等に割り当てられた確率を用いてスケジューリングを行なっている. しかし, 確率の見積りの悪さから, 演算コストの平均からの分散を求めるという意味での数学的な基礎づけがあるにも関わらず, 最適解を得られない例が報告されている. (→実験結果)

そこで, 今回我々は, 前章で定めた演算コストが各ステップで必要となる演算装置数の期待値を明確に反映しているという特徴を生かした, minimax 原理による選択法を用いたスケジューリング手法を提案する. 準備として, 前節で定めた演算コストを今, EPC (Exact-Probability oriented Cost) と呼び, ステップ S_j における値を EPC_j で表すこととする.

我々の目的は, 同種の演算を全ステップに渡って出来るだけ分散させて, 必要な演算数, すなわちスケジューリング終了時の $max_{j\in[1,m]}EPC_j$ の値を最小化することであり, 具体的には, EPC_j の値を評価するための手法が必要になる. そのための我々の基本的な方針は,

- 割り当てステップが確定していない演算に関して, 1ステップに割り当てた時に, 「効果」の高いものから優先的に割り当てていく.

ということであり, 「効果」を的確に表現するために演算コスト EPC を利用する. 以下, 手順を説明する.

4.2 スケジューリング手順

1. タイムフレームグラフを作成する.
2. 各演算 O_i をタイムフレーム内の各ステップ S_k に割り当てる試行を行い, 各試行時のグラフに対して,

$$EPC\,test_{i,k} := max_{j\in[1,m]}EPC_j$$

を求める.

3. 2 の $EPC\,test_{i,k}$ の内, $min_{i,k}\{EPC\,test_{i,k}\}$ となるような O_i の S_k への割り当てを実際に行なう.
4. タイムフレームグラフとクラスター情報を更新する.
5. 2 ～ 4 の操作を, 全ての演算の割り当てが完了するまで反復する.

手順3においては, $min_{i,k}\{EPC\,test_{i,k}\}$ となるものを「効果」最大としている. 図2の例に適用すると, M_3 を S_2 に割り当てる場合の $EPC\,test_{3,2} = \frac{7}{3}$ が最小であるので, M_3 を S_2 に実際に割り当てる. 続いて, $EPC\,test_{6,3} = 2$ が最小であるので, M_6 を S_3 に割り当て, スケジューリングが完了する.(図8)

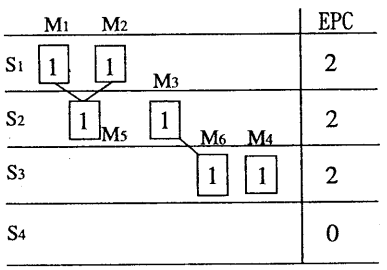

図 8: 図2の例に対する最終スケジューリング結果

4.3 計算量について

上記スケジューリングの計算量が $O(m^2n^2(n+\log m))$ であることが, 次のようにして分かる:

- 前述のスケジューリング手順での一回の反復で一つの演算の一ステップへの割り当てを行なうので, 多くても反復回数は n 回.
- 一つの反復の中で, 多くても n 個の割り当ての行なわれていない演算があり, それらをライフタイム中の各ステップへ割り当てる試行を行なうので, 試行回数は多くても mn 回である.
- 各試行の中では, 各ステップの演算コストの計算と, $max_{j\in[1,m]}EPC_j$ の計算を行なうので, 演算コストの計算量とソートに関する事実から, 計算量は, $O(mn) + O(m\log m) = O(mn + m\log m)$ である.

但し, ステップ数 m と演算数 n の大きさを比較すると, $m > n$ は, 演算等の実行をしていないステップがあることを意味しているので, 一般には $m \leq n$ と考えて良いと思われる. その時, 上記計算量は $O(m^2n^3)$ である. また, mn 回の試行は独立に処理できるので, 並列に処理することで処理時間を短縮することも可能である.

5 実験結果と考察

ここで述べたスケジューリング手法を, ベンチマークとして良く利用されている MAHA の例題と Elliptic Wave Filter に適用した結果を, いくつかの代表的なスケジューリング手法: MAHA[2], FDS[3], Improved FDS[4], [8] による結果と共に述べる.(図9)

MAHA の例題では, MAHA 以外は最適な結果を導いており, Elliptic Wave Filter では, FDS が 18 ステップで最適なスケジューリングを得られない他は, 本手法を含めて, 最適な結果が導かれている.

MAHAの例題				
step / 手法	4		8	
	+	-	+	-
MAHA	2	3	1	1
FDS	2	2	1	1
[8]	2	2	1	1
本手法	2	2	1	1

Elliptic Wave Filter								
step / 手法	17		18		19		21	
	*	+	*	+	*	+	*	+
FDS	3	3	2	3	2	2	1	2
Improved FDS	3	3	2	2	2	2	1	2
[8]	3	3	2	2	2	2	1	2
本手法	3	3	2	2	2	2	1	2

図 9: 実験結果

計算量については, Kernighan, Lin の min-cut algorithm を用いた [8] の手法が $O(mn^2 \log n)$ と最も小さい. FDS と Improved FDS の計算量は $O(m^2n^3)$ で, 上記の $m \leq n$ の仮定の下で, 本手法と同等である.

今回の実験で用いたグラフの中にも連続する同種演算は多く存在して, 本手法を効果的に利用することが出来た. 演算コストの構成の仕方からも判断できるように, 本手法は, 演算が連続して連なってクラスタリング出来れば出来るほど, 正確な確率に基づいた演算コストに近付き, 最終的なスケジューリング結果も, 最適になる可能性は高くなると考えられる.

上記 2 つの例題, あるいは他の小規模ないくつかのグラフに限れば, [8] は処理時間が短く, 全ての場合に最適解を得る結果となっており, その程度の規模のものに対して [8] は有効な手法と言える. しかし, ステップ数制約下での演算コスト最小化を目的とした手法の評価のために頻繁に使われているベンチマークとしては, これまで Elliptic Wave Filter が最も大きいというのが現状であるので, 本手法とそれら他手法のスケジューリング能力は, さらに大規模なグラフで, 比較, 検討していく必要がある.

6 まとめ

演算コスト最小化を目的にした, 新たな確率ベース演算コストと, その演算コストを用いた minimax 原理によるスケジューリング手法について述べた. 演算コストについては, まず理論的な基礎付けをした後, 一般のグラフに対する適用を試みた. その特徴をまとめると以下のようになる.

1. 従来のコスト計算法と同等の計算量で, かつ各ステップで必要となる演算装置数の期待値をより強く反映させることができる.

2. マルチサイクル化, パイプライン化, チェイン化された演算にも対応できる.

さらに, その演算コストを利用して, minimax 原理によるスケジューリング手法をとることで, 実用的な計算量で, 最適, または準最適なスケジューリングが得られることが確認された.

今後は, レジスタなど他のリソースコストへの本演算コストの適用範囲の拡大や, より規模の大きなグラフに対するスケジューリングの実験と評価などを進めていく予定である.

参考文献

[1] J. Lee, Y. Hsu, Y. Lin, "A New Integer Linear Programming Formulation for the Scheduling Problem in Data-path Synthesis," Proc. of the International Conf. on Computer-Aided Design, 1989, pp. 20–23.

[2] A.C. Parker, J.'T.' Pizarro, M. Mlinar, "MAHA: A Program for Datapath Synthesis," Proc. 23th Design Automation Conf., 1986, pp. 461–466.

[3] P.G. Paulin, J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans on Computor-Aided Design of Integrated Circuits and Systems, vol. 8, no. 6, June. 1989, pp. 661–679.

[4] W.F.J. Verhaegh, E.H.L. Aarts, J.H.M. Korst, P.E.R. Lippens, "Improved Force-Directed Scheduling," Proc. European Design Automation Conf., 1991, pp. 430–435.

[5] R. Dutta, J. Roy, R. Vemuri, "Distributed Design-Space Exploration for High-Level Synthesis System," Proc. 29th Design Automation Conf., 1992, pp. 644–650.

[6] R. Karri, A. Orailoglu, "Area-Efficient Fault Detection During Self-Recovering Microarchitecture Synthesis," Proc. 31th Design Automation Conf., 1994, pp. 552–556.

[7] A.W. Nagle, A.C. Parker,"Algorithms for multiple-criterion design of microprogrammed control hardware," Proc. 18th Design Automation Conf., 1981, pp. 486–493.

[8] In-Cheol Park, Chong-Min Kyung,"Fast and Near Optimal Scheduling in Automatic Data Path Synthesis," Proc. 28th Design Automation Conf., 1991, pp. 680–685.