設計自動化 81－10
(1996. 10. 18)

解空間の縮小を用いたスケジュール手法

平川裕介　　原嶋勝美　　福永邦雄

大阪府立大学工学部

〒593 大阪府堺市学園町1番1号

(0722)52-1161

E-mail:hirakawa@com.cs.osakafu-u.ac.jp

あらまし　本稿では，処理時間制約下での演算器数最小化スケジューリング手法を提案する．演算器数最小化スケジューリング手法では，演算は，割り当て可能な各制御ステップに対し，均等な割り当て可能性をもつと仮定することが多い．しかし，実際には各制御ステップで割り当て可能性が均等になるとは限らない．提案手法では，割り当て可能性に各制御ステップの割り当て可能な演算数に基づいた重みを与えることで，演算の分布を反映した演算器数の見積もりが可能となった．さらに，上記の見積もり方法を用いて，割り当て可能性の低い演算と制御ステップの組合せを逐次削除していくことで，大域的な演算器数の最小化を効率良く行なうことができた．

キーワード　高位合成，スケジューリング，タイムフレーム

A Time Contrainted Scheduling with the Time Frame Reduction

Yusuke Hirakawa　　Katumi Harashima　　Kunio Fukunaga

College of Engineering,Osaka Prefecture University

1-1,Gakuen-cho,Sakai,Osaka,593,Japan

(0722)52-1161

E-mail:hirakawa@com.cs.osakafu-u.ac.jp

Abstract

In this paper, we propose a time constrainted scheduling algorithm that minimizes hardware cost. Thought most approaches are to assume uniform probability of assigning an operation to any feasible control step, the assumption is not always realistic. Our approach estimates the number of operations with weighted probability reflecting the distribution of operations. It can also minimize the required number of functional units globaly and efficiently by combining the above estimation and the elimination of a pair of an operation and a control step with maximum assignment cost.

key words　high-level synthesis,scheduling,time frame

1 はじめに

LSI 自動設計の高位合成 (ハイレベルシンセシス) は, ハードウェアに依存しない動作記述からレジスタ転送レベルの動作記述を自動作成する技術である. 高位合成におけるスケジューリングは, 最終的に生成されるチップの遅延, 面積に多大な影響を与える重要な要素技術であり, 処理の並列性に着目し, 全処理時間あるいはハードウェアコストの最小化を目的として, 各演算の回路状態 (制御ステップ) への割り当てを行なう. ところが, これらの二つの目的は, 相反するため, 実際には一方を制約とし, 他方を最小化する手法がとられることが一般的である. 即ち, スケジューリングは,

1. 処理時間制約下でのハードウェアコスト最小化
2. ハードウェアコスト制約下での処理時間最小化

に大別できる [1]. これらは, いずれも NP 完全問題であり, 小規模な問題に対しては, 整数線形計画法 [2] を用いることで制約条件下での最適解を求めることはできるが, 問題規模の増加にともない計算時間が指数関数的に増加するため, 大規模な問題に対しては実用的ではない. このため, 多くの発見的な手法が提案されており, 代表的な処理時間制約下でのスケジューリングには, フォースディレクテッドスケジューリング法 [3][4], 繰り返し改善法 [5] などがある. これらは, 計算時間の面では改善はされたが, 解の最適性に問題が残っている.

本報告では, 処理時間制約下でのハードウェアコスト最小化を目的とするスケジューリング手法を提案する. ハードウェアコストとして, 各制御ステップで必要となる演算器数を見積もる場合, 演算の割り当て可能制御ステップへの割り当てに対し, 均等な割り当て可能性を与えることが一般的である. ところが, 均等な割り当て可能性が, 演算器数の少ないスケジューリングにつながるとは限らない. これに対し, 提案手法では, 演算の制御ステップに対する分布を反映した割り当て可能性を用いることで, 演算器数をより厳密に見積もっている. さらに, スケジューリング結果が, 局所最適解に陥ることを避けるために, 演算の割り当てを逐次決定していくのではなく, 演算の割り当てを行なわない制御ステップを逐次決定していくことで, 最終結果を得ている.

以下,2 章でスケジューリング問題の定式化を行ない, 3 章で提案手法について述べ,4 章で実験結果を示す.

2 スケジューリング

はじめに, 本節以下で用いる記号を定義する. 但し, 絶対値は集合を構成する要素数を表す.

O	:	全演算
		(=$\{ o_i \mid 0 < i \leq \lvert O \rvert \}$)
S	:	全制御ステップ
		(=$\{ s_i \mid 0 < i \leq \lvert S \rvert \}$)
T	:	全演算種
		(=$\{ t_i \mid 0 < i \leq \lvert T \rvert \}$)
C_t	:	演算種 t のコスト
type(o)	:	演算 o の演算種
asap(o)	:	演算 o が実行できる
		最も早い制御ステップ
alap(o)	:	演算 o が実行できる
		最も遅い制御ステップ
FU(s,t)	:	制御ステップ s で実行する
		演算種 t の演算数
FRAME(o)	:	演算 o が実行できる
		制御ステップ集合

本稿で対象とするスケジューリング問題は, データの依存関係を壊さずに, ハードウェアコストを最小にする演算の割り当てを決定することである. 提案手法では, ハードウェアコストは, 演算器のコストのみを考えている. 即ち, 目的であるハードウェアコストは次式となる.

$$Cost = \sum_{t \in T} C_t \times \max_{s \in S} FU(s,t)$$

従来の手法の多くは, 各制御ステップでの演算器数の見積もりのために次式の割り当て確率 ($Prob(o,s)$) を用いている.

$$\begin{aligned} Prob(o,s) &= \frac{1}{|FRAME(o)|} \\ &= \frac{1}{alap(o) - asap(o) + 1} \end{aligned}$$

$Prob(o,s)$ では, 各制御ステップへの割り当ては, 等確率で起きると仮定しているが, 実際には, データの依存関係のために等確率で起こ

るとは限らない. 例えば, 図 1に示すデータの依存関係のある演算 $o_1(FRAME(o_1) = \{s_1, s_2\})$,$o_2(FRAME(o_2) = \{s_2, s_3\})$ の割り当て確率は,

$$Prob(o_1, s_1) = \frac{1}{2}, \quad Prob(o_1, s_2) = \frac{1}{2},$$
$$Prob(o_2, s_2) = \frac{1}{2}, \quad Prob(o_2, s_3) = \frac{1}{2},$$

となるが, 実際の組合せ数は図 1に示す 3 通りが存在し, 正確な割り当て確率は,

$$Prob'(o_1, s_1) = \frac{2}{3}, \quad Prob'(o_1, s_2) = \frac{1}{3},$$
$$Prob'(o_2, s_2) = \frac{1}{3}, \quad Prob'(o_2, s_3) = \frac{2}{3},$$

となる. 上記の計算例からもわかるように $Prob(o, s)$ はデータの依存関係を考慮していないために, 正確な割り当て確率とはいえない. しかし, 正確な割り当て確率を計算するには, 多くの計算時間が必要であり, 実用的ではない. また, 総演算器数を最小にするためには, 各制御ステップで使用する演算器数を均等にする必要があるため, 多くの演算が割り当て可能である制御ステップにおいては, そうでない制御ステップに比べて割り当て可能性が低くなる. 例えば, 図 1において制御ステップ s_1に割り当てることができる演算数が, 制御ステップ s_2に割り当てることができる演算数より多い場合, 演算 o_1が, 制御ステップ s_1に割り当てられる可能性は低いと考えられる. そこで, 提案手法では, 従来の割り当て確率ではなく, 各制御ステップに割り当てができる演算の個数に基づいた存在確率を, 演算器数の見積もりに用いる.

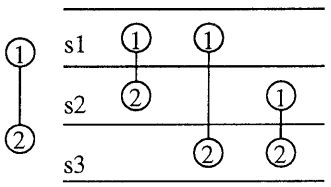

図 1: 演算の割り当て確率

3 提案手法

既に提案されている多くの手法は, ハードウェアコストの評価に基づき, 演算を実行する制御ステップを逐次決定する. しかし, 未割り当ての演算が多いときは, 正確なコスト評価が困難であるため, 決定される割り当てが最適解とは限らない. そこで, 提案手法では, 処理の初期段階から割り当てを行なわないように, 解を悪化させる可能性の高い演算と制御ステップの組合せを逐次削除することで, 最適解となる可能性の高い組合せを絞り込んでいく.

演算 o が, 制御ステップ s で実行されない場合としては,

(A1) 演算数が制御ステップ s より少ない他の制御ステップに割り当てることができる,

(A2) 他の演算の割り当てにより制御ステップ s での実行が不可能になる,

が挙げられる. そこで, 提案手法では, はじめに, 後述する (A1) を反映した存在確率を用いて, 各制御ステップの演算器数を見積もり, 最大の演算器数となる制御ステップ集合 (S_{can}) を得る. つぎに, 各制御ステップ $s(s \in S_{can})$ において実行可能な演算から, 演算器数の見積もり値が小さい他の制御ステップに割り当てることのできる演算 o を得る. さらに, データ依存関係のある他の演算への影響 (A2) を考慮するために, 演算 o の制御ステップ s での実行を禁止した場合の演算器数の見積もり値の変化により削除するか否かを決定する.

提案手法のアルゴリズムを図 2に示す. 制約として, 処理時間 (制御ステップ数,$|S|$) を与えると, 各演算の実行可能制御ステップは, ASAPS(as soon as possilbe scheduling),ALAPS(as late as possible scheduling) により求められる. 以下, 処理手順に沿って説明する.

```
0 前処理 (FRAME(o) ⇐ ASAPS,ALAPS )
1 while(!全演算をスケジューリングしたか){
2     各演算の存在確率を計算
3     演算器数の見積もり値を計算
4     削除対象を選択
5     各削除対象を評価
6     最良の削除を実行する
7 }
```

図 2: アルゴリズム

3.1 存在確率

存在確率 (P(o,s)) は, 演算 o の制御ステップ $s(s \in FRAME(o))$ への割り当て可能性を示す. 例えば,$FRAME(o_1) = \{s_1, s_2\}$ であり, 制御ステップ s_1,s_2にはそれぞれ 3 個および 5 個の演算が実行可能とする. 総演算器数を最小にするためには, 各制御ステップでできるだけ一様数の演算器を使用しなけらばならないため, データ依存関係を考慮しなければ, 演算 o_1は, 制御ステップ s_1に割り当てることが望ましい. 従って, 制御ステップ s_1への割り当て可能性は, 制御ステップ s_2に対するものより大きくする $(P(o_1, s_1) > P(o_1, s_2))$. 以下, この考えをもとに存在確率 $(P(o, s))$ を定義する.

1. 多くの演算が割り当て可能な制御ステップ s で実行可能な演算は, 制御ステップ s に割り当てられる可能性は小さい. 従って, 存在確率は, 制御ステップに割り当て可能な演算器数に反比例するように定義する. 但し,$MAX\_FU(s,t)$ は, 制御ステップ s への割り当て可能な演算種 t の演算数を表す.

$$p(o,s) = \frac{1}{MAX\_FU(s, type(o))}$$

2. 自由度 (=$|FLAME(o)|$) の大きい演算は, 自由度の小さい演算より 1 つの制御ステップ当たりの割り当て可能性は小さい. また, 各演算の演算器数の見積もり値に対する影響を同等にするために, $p(o,s)$ を次のように正規化する.

$$P(o,s) = \frac{p(o,s)}{total(o)}$$

$$\sum_{s \in FRAME(o)} P(o,s) = 1$$

$$total(o) = \sum_{s \in FRAME(o)} p(o,s)$$

図 3の演算 o_1,o_2の存在確率の計算例を以下に示す. 同図左は, 各演算の FRAME を表し, 同図右では, 矩形の幅が各演算の存在確率を表す.

$$\begin{aligned} total(o_1) &= \tfrac{1}{2} + \tfrac{1}{5} + \tfrac{1}{2} \\ total(o_2) &= \tfrac{1}{2} + \tfrac{1}{5} \end{aligned}$$

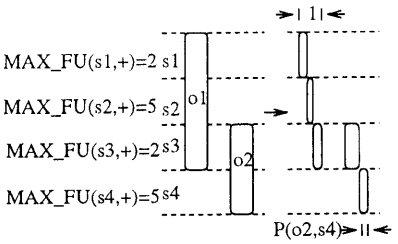

図 3: 演算の存在確率

$$\begin{aligned} P(o_1, s_1) &= \frac{\frac{1}{2}}{total(o_1)} = \frac{5}{12} \\ P(o_1, s_2) &= \frac{\frac{1}{5}}{total(o_1)} = \frac{2}{12} \\ P(o_1, s_3) &= \frac{\frac{1}{2}}{total(o_1)} = \frac{5}{12} \\ P(o_2, s_3) &= \frac{\frac{1}{2}}{total(o_2)} = \frac{5}{7} \\ P(o_2, s_4) &= \frac{\frac{1}{5}}{total(o_2)} = \frac{2}{7} \end{aligned}$$

3.2 演算器数の見積もり

前述した存在確率は, 演算の各制御ステップへの割り当ての可能性を表している. 従って, 各制御ステップでのそれらの和を演算器数の見積もり値 $(EFU(t,s))$ とする.

$$EFU(t,s) = \sum_{o \in \{s \text{ で実行可能な演算種 } t \text{ の演算 }\}} P(o,s)$$

例えば, 図 4に示すデータフローグラフ (Data Flow Graph,DFG) に対し, 各演算の FRAME(o) は図 5で表され, 各ノードの存在確率は図 6で表されているものとする. 但し, 図 6において各ノードに対応する矩形の幅が存在確率を表す. この結果から, 演算器数の見積もり値 $(EFU(t,s))$ は,

$$\begin{aligned} EFU(+,s_1) &= 2 + \tfrac{4}{10} \\ EFU(+,s_2) &= 1 + \tfrac{3}{10} + \tfrac{1}{2} + \tfrac{1}{4} \\ EFU(+,s_3) &= 1 + \tfrac{3}{10} + \tfrac{1}{2} + \tfrac{1}{4} \\ EFU(+,s_4) &= 1 + \tfrac{2}{4} \end{aligned}$$

となる (図 7).

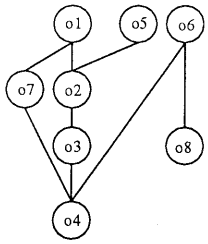

図 4: Data Flow Graph

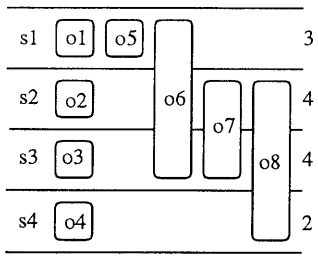

図 5: 演算の割り当て可能範囲

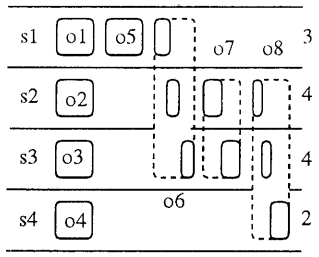

図 6: 演算の存在確率

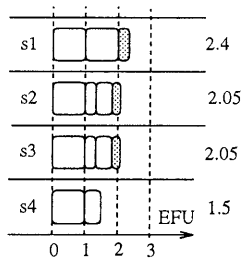

図 7: 演算器数の見積もり

3.3 削除

3.3.1 削除対象

存在確率は,(A1) のみを反映している. しかし, 演算の割り当ては, 前述したようにデータ依存関係のある演算の実行する制御ステップによっても制約を受ける. そこで, 演算の制御ステップへの割り当てを削除した DFG を評価することで, 削除による他の制御ステップの演算器数への影響 (A2) を調べる. しかし, 全ての可能性を試すことは, 計算時間的に困難であるため, 演算器数の見積もり値が大きく, 他の制御ステップへの割り当てが望ましい演算, 即ち, 存在確率が小さい演算を削除対象 ($Candidate$) として選択する. 但し, 実際には, 演算器数の見積もり値が小数の場合には, 小数部分を切り上げた数を必要演算器数として, 削除対象を選択している. 選択手順を図 8に示す. ここで, 関数 $ceil(x)$ は, 実数 x の切り上げた整数を返す関数, 関数 $get\_min(s,t)$ は, 制御ステップ s において実行可能な演算種 t の演算集合から存在確率が最小である演算を返す関数である. 例えば, 図 7においては, 必要演算器数は 3,$Candidate = \{(o_6, s_1), (o_8, s_2), (o_8, s_3)\}$ となる.

3.3.2 削除

前節で求めた各削除候補 ($Candidate$) に対して削除を行なった場合の演算器数の見積もり値 (EFU'(t,s)) を計算する. 削除手順を図 9 に示す. 演算の割り当てを禁止し, その演算の ASAP,ALAP 値が変化すると, その演算とデー

$max = \max_{t \in T, s \in S} ceil(EFU(t,s))$
for t ∈ T , s ∈ S {
  if $(ceil(EFU(t,s)) = max)$ {
    $op \Leftarrow get\_min(s,t)$
    $Candidate \Leftarrow (op,s)$
  }
}

図 8: 削除対象の選択

```
for (o, s) ∈ Candidate {
    (o,s) を削除 ( delete(o,s) )
    存在確率計算
    演算器数の見積もり値計算
    評価計算
}
```

図 9: 削除のアルゴリズム

タの依存関係がある演算の ASAP,ALAP 値も変化する. この処理を行なうのが, 関数 $delete(o,s)$ である. また, 削除や ASAP,ALAP 値が変化することで, 各制御ステップに割り当て可能な演算数 $(MAX\_FU(s,t))$ も変化するため, データの依存関係のない演算も存在確率の再計算が必要である.

図 6の演算 o_6の制御ステップ s_1への割り当てを禁止した場合の, 各演算の FRAME と存在確率を図 10に示す. 演算 o_6とデータの依存関係のある演算 o_8は, 演算 o_6の ASAP 値の変化により,ASAP 値が変化している. また, 演算 o_7は, 演算 o_6とデータの依存関係はないが, $MAX\_FU(s_1,+), MAX\_FU(s_2,+)$ の変化のために存在確率が変化している.

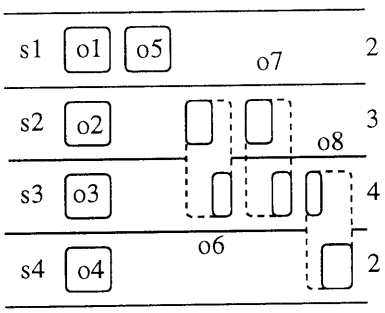

図 10: 演算の削除例

3.4 評価

提案手法では,FDS と同様に演算器数の見積もり値 $(EFU(t,s))$ の変化によって, 削除の評価 $(Pri(o,s))$ を行なう. 評価式は, 各制御ステップの演算器数の見積り値ができるだけ一様に近づくように,

- 演算器数の見積もり値の大きい制御ステップでの EFU の減少が, 演算器数の見積もり値の小さい制御ステップでの減少より優先する,
- 演算器数の見積もり値の小さい制御ステップでの EFU の増加は, 演算器数の見積もり値の大きい制御ステップでの増加より優先する,

の性質をもつように次式のように定義する.

$$\begin{aligned} Pri(o,s) &= \sum_{s\in S, t\in T} EFU(t,s) \\ &\times \{EFU'(t,s) - EFU(t,s)\} \end{aligned}$$

例えば, 削除 (o_6,s_1) の評価は, 図 6と図 10から

$$\begin{aligned} Pri(o_6,s_1) &= 2.4 \times \{2 - 2.4\} \\ &+ 2.05 \times \{1\tfrac{8}{7} - 2.05\} \\ &+ 2.05 \times \{1\tfrac{31}{21} - 2.05\} \\ &+ 1.5 \times \{1\tfrac{2}{3} - 1.5\} \end{aligned}$$

となる. 上式を用いて, 全ての削除候補に対し, 評価を行ない, 最小の評価の削除のみを採用する.

3.5 計算量

本節では, 提案手法のアルゴリズム (図 2) の計算量を示す. 但し, 演算数 $n(=|O|)$, 制御ステップ数 $m(=|S|)$ とする. 以下, 各説明は, 図 2の行番号に従っている.

(1) 演算 o と演算 o が実行できる制御ステップの組合わせは, 最大で $(m \times n)$ 組ある. 1回の loop で, 少なくとも一組を削除するため, 最大で $(m \times n)$ 回 loop を繰り返すことになる (計算量 $O(mn)$).

(2a) 各演算に対し, 各制御ステップに割り当て可能か否かを調べ, $MAX\_FU(s,t)$ を求める (計算量 $O(nm)$).

(2b) 各演算に対し, 存在確率の総和を求め, 正規化する (計算量 $O(nm)$).

(3) 各制御ステップ, 各演算種ごとに存在確率の総和を求める. 各制御ステップには, 最大で n 個の演算が存在する (計算量 $O(mn)$).

(4a) 制御ステップごとの演算器数の見積もり値から, 削除候補ステップ集合 (S_{can}) を選択する (計算量 $O(m)$).

(4b) 削除候補ステップ集合の各要素 (要素数の最大は m) に対し, 最小の存在確率をもつ演算を選択する (計算量 $O(mn)$).

(5) 削除候補に対し, 評価を計算する (候補数は $|S_{can}|(\leq m)$).

  (a) 削除による他の演算への影響を最大で $(n-1)$ 個の演算に対して調べる (計算量 $O(n)$).

  (b) 各候補に (2) から (4) の操作を行なう (計算量 $O(mn)$).

  (c) 評価を行なう (計算量 $O(m)$).

(6) 削除による他の演算への影響を最大で $(n-1)$ 個の演算に対して調べる (計算量 $O(n)$).

以上より, 計算量は,$O(n^2m^3)$ となる.

4 実験

本節では, はじめに存在確率の有効性を示す. 次に, 提案手法での実験結果を示す. 以下の実験では, 演算器コストは, $C_+ = C_\times$ =1 とする.

4.1 存在確率

2 章で述べたように, 従来提案されている多くの手法は, 演算の各制御ステップへの割り当ての可能性を均等と考え, 割り当て確率 ($Prob(o,s)$) を用いて, 演算器数の見積もり値を計算している. その評価値を使用し, 準最適解を得ることができる手法にフォースディレクテッドスケジューリング法 (FDS) がある. 従来の割り当て確率と存在確率を比較するために, FDS の割り当て確率のみを提案手法の存在確率に置き換えた手法 (IMFDS) を用いて実験を行なった. 入力データは, 乱数により作成し, 実験は, ノード数 60,125,226,331,571 のデータを各 20 個ずつ計 100 個のデータに対して行なった.

表 1に実験結果を示す. 但し, 最小解は, 最小の演算器数 (=演算数/制御ステップ数) となったものである. 表 2に結果の一部を示す. 括弧内の数字は,FDS との演算器数の差である. 実験から, 提案手法の存在確率 $P(o,s)$ を使用することで, FDS では, 最適解が得られなかった入力データに対しても最適解が得られることが確認できた. この結果から, 存在確率が演算器数の見積もりに有効であることがわかる. また,FDS の計算量が,$O(m^2n^3)$(制御ステップ数 m, ノード数 n) に対し,IMFDS の計算量は, 存在確率の計算が, 割り当て確率の m 倍となるため $O(m^3n^3)$ となるが, 実際には, 制御ステップ数が 10〜40 の場合でも, 計算時間は 1〜2.5 倍にとどまった.

表 1: 存在確率の有効性

改善データ	FDS と同等な結果	最小解
13/100	79/100	72/100

表 2: 存在確率の有効性の結果例

ノード数	FDS		IMFDS		処理時間 $\frac{IMFDS}{FDS}$
	×	+	×	+	
60	3	4	2(-1)	4(0)	2.11
60	2	4	2(0)	4(0)	1.44
125	5	6	5(0)	6(0)	1.51
125	5	5	4(-1)	5(0)	1.57
125	5	5	5(0)	6(1)	1.70
226	6	10	6(0)	10(0)	2.25
226	6	11	6(0)	11(0)	1.93
331	12	13	12(0)	12(-1)	2.15
331	10	10	10(0)	11(1)	2.00
571	16	23	16(0)	22(-1)	2.35
571	18	25	17(-1)	25(0)	1.96
571	15	21	15(0)	21(0)	2.30

4.2 提案手法

提案手法を複数の対象を用いて実験を行なった. 入力データは, 乱数により作成し, 実験は, ノード数 60,125,226,331,571 のデータを各 20 個ずつ計 100 個のデータに対して行なった.

実験結果を表 3に示す. 但し, 最小解は, 最小の演算器数 (=演算数/制御ステップ数) となったものである. 結果の一部を表 4に示す. 括弧内の数字は,FDS との演算器数の差である. 提案手法は, 半数以上の入力データに対して準最適解が得られる FDS と同等, あるいはそれ以上の結果が得られた. 演算器数が増加する場合でも, ほとんどが 1 個の増加, 最悪でも (ノード数 571) でも, 総演算器数 36 個に対して,3 個の演算器の増加となった. また, データの規模が増加するのにともない FDS との処理時間の比は, 小さくなる傾向があった. これは, 一般に $n > m$(ノード数 n, 制御ステップ数 m) であり, FDS の計算量が,$O(m^2n^3)$ に対し, 提案手法の計算量は,$O(m^3n^2)$ であること, n が定数であることに対し, 実際には, m は各ノードの自由度 $(= |FRAME(o)|)$ であり, ノード数が大きいほど $(m - |FRAME(o)|)$ が大きくなることによると考えられる. これらの結果から, 提案手法は, 効率良く準最適解が得られることがわかる.

表 3: 提案手法の結果

改善データ	FDS と同等な結果	最小解
9/100	60/100	51/100

表 4: 提案手法の結果例

ノード数	FDS		ours		処理時間 $\frac{ours}{FDS}$
	×	+	×	+	
60	2	3	2 (0)	3 (0)	1.13
60	3	5	2 (-1)	4 (-1)	0.98
125	4	5	4 (0)	5 (0)	0.43
125	4	5	4 (0)	6 (1)	0.28
226	5	9	5 (0)	9 (0)	0.20
226	5	9	5 (0)	8 (-1)	0.32
331	10	10	10 (0)	10 (0)	0.27
331	9	9	10 (1)	10 (1)	0.96
571	14	19	14 (0)	20 (1)	0.10
571	15	21	15 (0)	21 (0)	0.10
571	14	19	16 (2)	20 (1)	0.11

5 むすび

本報告では, 演算器数の最小化を目的とした時間制約下でのスケジューリング手法を提案した. 提案手法は, 演算の割り当て分布を反映した新たな演算器数の見積もり方法を用いることおよび演算を実行しない回路状態を逐次決定することにより, 最適解に近い解を探索していることを示した. また, 乱数により作成した複数のデータに対して実験を行ない提案手法の有効性を確認した. 今後の課題としては, 効率の良い削除対象の選択方法, レジスタ, バスなどの他のハードウェアコストの考慮などを考えている.

参考文献

[1] D.Gajski,N.Dutt,A.Wu and S.Lin, High-Level Synthesis:Introduction to Chip and System Design, Kluwer Academic Publishers,1992.

[2] J.Lee,Y.Hsu and 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.

[3] P.G.Paulin and J.P.Knight, "Force Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans on Computer-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 and P.E.R.Lippens, "Improved Force-Directed Scheduling," Proc. of the European Design Automation Conf. ,1991,pp.430-435.

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