分枝限定に基づく最適解を保証するリソースバインディング手法

中本真児† 戸川望†
柳澤政生† 大附辰夫†

本論文では，ディジタル信号処理ハードウェアのデータパス設計を対象としたリソースバインディング手法を提案する．提案手法は分枝限定法により，接続コストを最小化するようにデータフローグラフ内の演算を演算器に，演算結果データをレジスタに，データ転送をデータ転送路に割り当てる．提案手法は直接，分枝限定法をリソースバインディング問題に適用したものである．従来のようにリソースバインディング問題を演算器割当て，レジスタ割当て，データ転送路割当てに分割することなく，1つの問題として扱うことが可能となり，現実的な時間で最適解を得ることができる．さらに分枝限定法を適用する際に，前処理として発見的手法により解空間を削減し高速に準最適解を算出することができる．計算機実験により，リソースバインディングの典型的な例題に対して数秒から4日程度で最適解を得ることを確認した．

An Optimal Binding Algorithm in High-level Synthesis System for Digital Signal Processing

Shinji Nakamoto,† Nozomu Togawa,† Masao Yanagisawa† and Tatsuo Ohtsuki†

This paper proposes an optimal resource binding algorithm in the high-level synthesis system for digital signal processing. The algorithm is based on the branch-and-bound method and assigns operations, variables and data transfers in given data-flow graph to functional units, registers and data path so as to minimize the costs for interconnections. Since the branch-and-bound method is directly applied to a resource binding problem, the proposed algorithm can provide an optimal solution without dividing the resource binding problem into three subtasks, such as functional unit, register and data path binding. Moreover, before applying the branch-and-bound method, the algorithm reduces the design space by using a heuristic algorithm, and obtains near-optimal solutions. Experimental results demonstrate effectiveness and efficiency of the algorithm.

1. はじめに

ディジタルシステムによるデータ処理は，信号処理に代表されるデータの流れが主体の処理と伝送処理に代表されるデータの制御が主体の処理とに大別される．データの流れが主体の処理は，連続したビット幅固定の通信データに対し，加算や乗算といった演算を繰り返し実行するもので，ディジタルフィルタ，離散コサイン変換処理，数値微積分のハードウェア解法などの処理である．データの制御が主体の処理は，データからの特定のビット列の検出，タイミング情報の抽出といったデータの制御を実行するもので，プロトコル処理，画像処理におけるエントロピー符号化等の処理である．

上述の処理を専用のディジタル信号処理ハードウェアで実現する場合，データの流れが主体の処理は複雑なデータパス系を必要とする．このときデータパス系の設計自動化を実現する高位合成システムには，次の2つの要件を満たすべきであると考える．

- 設計者によって与えられた動作仕様を満たす複数の設計候補を提供し，設計者の判断により最適設計が選択可能である．
- 合成された設計候補に対する性能，性質に関する多くの情報を設計者に提供する．

このような考えに基づき我々は，ディジタル信号処理ハードウェアのデータパス設計を対象とした高位合成システムを構築している[12]．本システムは主に，(1)

† 早稲田大学理工学部電子・情報通信学科
Department of Electronics, Information and Communication Engineering, Waseda University

データフローグラフ（DFG）ジェネレータ，(2) スケジューラ，(3) リソースバインダ，(4) ハードウェア記述言語（HDL）ジェネレータから構成される．本システムの特徴は次の点である．入力となる動作記述からDFG ジェネレータにより内部表現となる DFG を複数個生成する．次に各 DFG に対してスケジューリング，リソースバインディングを高速に実行することで具体的な評価指標を設計者に与える．これらは，入力となる動作記述に対するデータパスの設計候補となる．設計者は最適と判断した設計候補に対応する DFG に対して最適なスケジューリング，リソースバインディングを実行して最終的な HDL 記述を得る．

いま，上述の最適なリソースバインディングに焦点を当てる．リソースバインディング手法に関する研究はこれまで多くの報告がなされている．既存のリソースバインディング手法は，発見的算法に基づく手法[2),8),9),14)]と最適解を保証する手法[1),5)~7),10),11),13)]に大別される．発見的算法に基づく手法には，クリーク分割に基づく手法[9),14)]，2 部グラフの最小重みマッチングに基づく手法[2)]などがある．最適解を保証するリソースバインディング手法では，最適解をいかに現実的な時間で算出するかが重要となる．従来，最適解を保証するリソースバインディング手法は，リソースバインディング問題を整数線形計画問題に変換し，分枝限定法[5),7),10),13)]，シミュレーテッド・アニーリング法[1)]などを適用しているが，どの手法においても問題の規模が大きくなると計算時間が増大し最適解を求めることができなくなるため，何らかの前提条件を設けて問題の複雑度を緩和する方針をとっている．たとえば，文献 7) ではコントロールステップで入力となる DFG を分割し問題の規模を抑えている．また文献 5), 13) では，リソースバインディング問題を，演算器割当て問題，レジスタ割当て問題，データ転送路割当て問題に分割し，それらを逐次的に解くことで問題の複雑度を緩和し計算時間の増大を防いでいる．

本論文では，ディジタル信号処理ハードウェアのデータパス設計を対象とした最適解を保証するリソースバインディング手法を提案する．提案手法は，スケジューリングされた DFG，リソース（演算器の種類やその数，レジスタ数）が与えられたときに，DFG 内の演算を演算器に，演算結果データをレジスタに，データ転送をデータ転送路に割り当て，接続コストが最小となるレジスタトランスファレベルのデータパスを合成するものである（接続コストは 2 章で定義する）．提案手法の基本アルゴリズムは，初期割当てと分枝限定法による解空間の探索から構成される．初期割当てでは，解の最適性を損なわないように演算器，レジスタ，データ転送路割当てをし，後段に当たる分枝限定法が探索すべき解空間の削減を実現する．続いて分枝限定法を用いて演算器割当て，レジスタ割当ておよびデータ転送路割当てを実現する．通常，組合せ最適化問題を分枝限定法により解決する場合，問題をまず整数線形計画問題に変換したのち分枝限定法が適用される[3)]．ところが提案手法では，分枝限定法を直接リソースバインディング問題に適用するため，従来のようにリソースバインディング問題を演算器割当て，レジスタ割当て，データ転送路割当てに分割することなく，1 つの問題として扱うことが可能となり，しかも現実的な時間で最適解を得ることができる．さらに初期割当ての際に，発見的手法による演算器割当て，レジスタ割当てをあらかじめ済ませることで，後に続く分枝限定法の探索空間を削減することができ，その結果として高速に準最適解の算出を可能とする．計算機実験により，提案手法の有効性を確認する．

本論文は次のように構成される．2 章では，本論文で扱うリソースバインディング問題を定義する．3 章では，分枝限定に基づく最適解を保証するリソースバインディング手法を提案する．4 章では，計算機実験結果を示し，提案手法の有効性を確認する．5 章では，本論文をまとめる．

2. 準　　備

2.1 データフローグラフ

構築している高位合成システムでは入力となる動作記述を内部表現であるデータフローグラフ（DFG: Data-Flow Graph）に変換する．DFG は有向グラフ $G(V,E)$ で表現される．ここで $V=\{v_1,v_2,\cdots,v_n\}$ は演算に相当する節点集合，E は各演算間のデータ依存関係を表す枝集合であり，枝 $e_{i,j}\in E$ は節点 v_i から節点 v_j $(v_i,v_j\in V)$ にデータの流れが存在することを表す．また枝 $e_{i,j}$ は節点 v_j が何イタレーション前の節点 v_i の結果に依存するかを表す数値を持ち $deg(e_{i,j})$ で示される．各節点は演算タイプを持つ．演算タイプの集合を OT とする．in，out で表される外部入力と外部出力も演算の一種として考え，演算タイプは $in,out\in OT$ とする．**図 1** は DFG の例である．節点内の数字は演算番号を表し，算術記号はその演算のタイプを表す．

2.2 アーキテクチャモデル

図 2 に本システムが対象とするアーキテクチャモデルの例を示す．データパスは演算器，記憶素子，データ転送路および結線からなる．使用可能な演算器は加

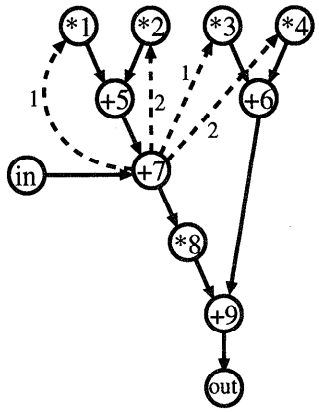


図1 DFG
Fig. 1 DFG.

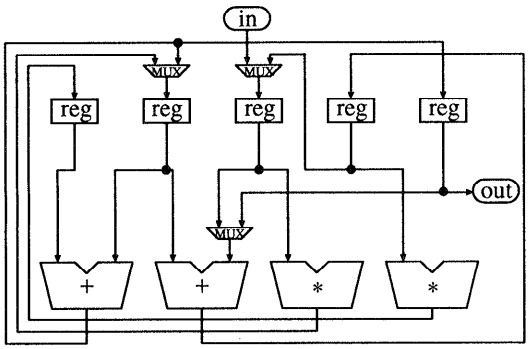


図2 アーキテクチャモデルの例
Fig. 2 An exapmle of the architecture model.

減算器，乗算器，除算器，比較器があり，いずれも2入力1出力の端子を持つ．DFGの節点すなわち演算は，演算器によって実行される．DFGの各節点が持つ演算タイプ $ot \in OT$ に1対1対応する演算タイプを演算器も持ち，各演算タイプ $ot \in OT$ に対して，そのタイプの演算を実行する演算器が1種類だけ存在するとする．

演算タイプ $ot \in OT$ の演算器に対し，演算に必要なクロックサイクル数を $ct(ot)$，データの入力間隔を $ii(ot)$ とする．演算タイプ ot に対して $ct(ot) \geq 2$ なる演算器はマルチサイクル演算器であり，演算の実行に複数サイクルを要する．このような演算器のうち，$ct(ot) = ii(ot)$ なる演算器は非パイプライン演算器であり，$ct(ot) > ii(ot)$ なる演算器はパイプライン演算器である．非パイプライン演算器は，演算途中のサイクルに次のデータを入力することができない．パイプライン演算器は，演算器内部にパイプラインレジスタを持ち，演算途中のサイクルで次のデータを入力することができる☆．記憶素子として1入力1出力の端子を持つレジスタがある．データ転送路には接続線とマルチプレクサ（MUX）が使用される．マルチプレクサは各演算器，レジスタの同一端子に2本以上の接続線が入力される場合に使用され，データはマルチプレクサを介して演算器またはレジスタに入力される．接続線は演算器–マルチプレクサ，演算器–レジスタ，レジスタ–マルチプレクサおよびレジスタ–レジスタ☆☆間にのみ存在し，演算器–演算器およびマルチプレクサ–マルチプレクサ間には存在しない．また外部にデータを入出力するポートとして1出力端子を持つ入力ポート，1入力端子を持つ出力ポートがある．

2.3 リソースバインディング

状態とは，クロックに同期したタイムステップであり，1状態は1クロック周期において実行される．状態の集合を $ST = \{i \mid 1 \leq i \leq CT\}$ とする．CT（コンピュテーションタイム）は与えられたDFGを1イタレーション実行するのに要するクロック数である．外部入力に入力されるデータの間隔を II（入力間隔）としたとき，同一クロックに実行される状態の集合 $CS(i)$ $(1 \leq i \leq II)$ を，$CS(i) = \{j \in ST \mid i \equiv j \pmod{II}\}$ とする．$CS(i)$ はコントロールステップと呼ばれ制御回路の一状態を表す．

DFG $G(V,E)$ のスケジューリングとは，$v \in V$ の表す演算に対し，その演算が開始される状態 $\sigma(v)$ $(1 \leq \sigma(v) \leq CT)$ を決定することである．節点 v は状態 $\sigma(v)$ に割り当てられたという．**図3** は図1のDFGを $CT = 9$，$II = 4$ としてスケジューリングしたものであり，そのうちの2イタレーション分を表している．$II = 4$ であることから状態4と状態8に割り当てられた演算 “+7” と “+9” は同時に実行される．演算 “+7” から演算 “*1” と “*3” に向かうイタレーションを跨ぐ枝を $e_{7,1}, e_{7,3}$ と書くとき $e_{7,1}, e_{7,3}$ は $deg(e_{7,1}) = deg(e_{7,3}) = 1$ である．

演算器の集合を FU，レジスタの集合を R，データ転送路の集合を DT とする．DFG $G(V,E)$ のリソースバインディングとは，スケジューリングの際に $\sigma(v)$ $(v \in V)$ が決定したDFG $G(V,E)$ に対し，(1) $v \in V$ が表す演算を実行する演算器 $fu \in FU$，(2)

☆ 本手法では現在のところ，ALUのような複数の演算を実行できるような演算器には対応していないが，複数の演算を表す演算 $alu \in OT$，演算 alu を実行することができる演算器 fu_{alu} を考え，複数の演算の実行サイクル数がすべて等しいと仮定することでALUのような複数の演算を実行できるような演算器に対応することが可能である．

☆☆ レジスタ–レジスタ間の接続線は，3.2.2項で後述する変数を分割する場合のみに限定している．

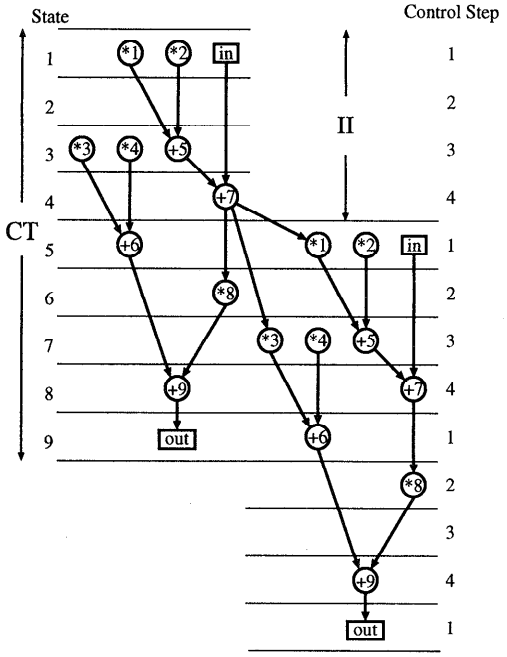


図3 スケジューリング後の DFG
Fig. 3 Scheduled DFG.

$v \in V$ が表す演算の演算結果データを保存するレジスタ $r \in R$, (3) $e_{i,j} \in E$ が表すデータ転送を実行するデータ転送路 $dt \in DT$ を決定し，レジスタトランスファレベルのアーキテクチャを合成することである．(1), (2), (3) を決定することをそれぞれ演算器割当て，レジスタ割当て，データ転送路割当てと呼ぶ．演算器割当て，レジスタ割当て，データ転送路割当ては次のように定義される．

2.3.1 演算器割当て

DFG の節点 $v \in V$ の表す演算 op は（以後節点 v と演算 op は同意で用いる），演算 op の演算タイプを ot とするとき，

$$CS_a(op) = \bigcup_{\sigma(op) \le i \le \sigma(op)+ii(ot)-1} CS(i)$$

において演算が実行される演算器を占有する．演算器 fu にすでに割り当てられた演算集合を OP_{fu} とするとき，演算 op を演算器 fu に割当て可能とは，op と fu の演算タイプが等しく，$\forall op_{fu} \in OP_{fu}$ に対して $CS_a(op) \cap CS_a(op_{fu}) = \emptyset$ の場合のときをいう．演算 op を演算器 fu に割当て可能なときのみ op を fu に割り当てることができ，op を fu に割り当てたとき op は $CS_a(op)$ において fu を占有する．

2.3.2 レジスタ割当て

DFG $G(V,E)$ がスケジューリングされ，節点 $v \in V$ が割り当てられた状態 $\sigma(v)$ $(v \in V)$ が決定すると，節点 v が表す演算の演算結果データ（以後変数と呼ぶ）が演算によって出力されてから他の演算の入力となるまでの時間間隔が決定する．この時間間隔を生存区間と呼び，次のように定義する．DFG $G(V,E)$ の節点 $v_i \in V$ に対し，節点 v_i が表す演算タイプを $ot \in OT$，節点 v_i の出枝の集合を $E_o(v_i)$ とする．枝 $e_{i,j} \in E_o(v_i)$ の生存区間を状態集合 $ST_{lt}(e_{i,j}) = \{lt \mid \sigma(v_i) + ct(ot) \le lt \le \sigma(v_j) + II \cdot deg(e_{i,j})\}$ で定義する．状態 $s(e_{i,j}) = \sigma(v_i) + ct(ot)$ を枝 $e_{i,j}$ の生存区間の始点，状態 $t(e_{i,j}) = \sigma(v_j) + II \cdot deg(e_{i,j})$ を枝 $e_{i,j}$ の生存区間の終点と呼ぶ．値 $l(e_{i,j}) = t(e_{i,j}) - s(e_{i,j}) + 1$ を枝 $e_{i,j}$ の生存区間長と呼ぶ．節点 v_i が出力する変数 vr の生存区間 $LT(vr)$ は，v_i の出枝の中で生存区間長が最大の枝 e_l の生存区間 $ST_{lt}(e_l)$ で定義する．変数 vr_i, vr_j において $LT(vr_i) \cap LT(vr_j) \neq \emptyset$ が成立するとき，変数 vr_i, vr_j は生存区間が重複するという．変数 vr の生存区間を図で表現する場合，生存区間の始点から生存区間の終点へ向かう矢印で表現する．変数の生存区間の例を**図4**に示す．図4(a) はスケジューリング後の DFG の例で，節点 v_1 から節点 v_2, v_3, v_4 に出枝が出ている．このとき枝 $e_{1,2}$ の生存区間の始点は状態 $s(e_{1,2}) = 2$ であり，生存区間の終点は状態 $t(e_{1,2}) = 3$ である．$e_{1,2}$ の生存区間長は $l(e_{1,2}) = 2$ である．同様にして $s(e_{1,3}) = s(e_{1,4}) = 2$, $t(e_{1,3}) = 4$, $t(e_{1,4}) = 2$, $l(e_{1,3}) = 3$, $l(e_{1,4}) = 1$ となる．このとき生存区間長が最大の v_1 の出枝は $e_{1,3}$ であるので，節点 v_1 が出力する変数を vr_1 としたとき変数 vr_1 の生存区間は $LT(vr_1) = \{2,3,4\}$ となる．

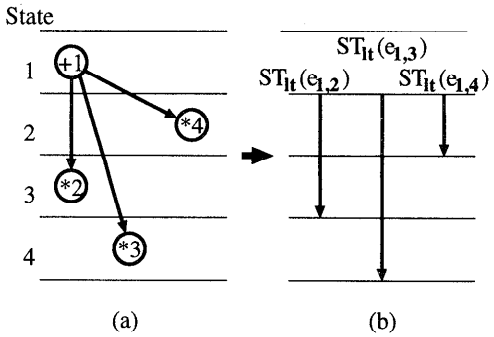


図4 変数の生存区間．(a) スケジューリング後の DFG．(b) 節点 v_1 の出枝の生存区間
Fig. 4 Lifetime of a variable. (a) Scheduled DFG. (b) Lifetime of the outgoing edges of the node v_1.

演算の場合と同様にして，変数 vr がレジスタを占有するコントロールステップの集合を $CS_a(vr)$ とする．レジスタ r にすでに割り当てられている変数集合を VR_r とするとき，変数 vr をレジスタ r に割当て可能

とは，$\forall vr_r \in VR_r$ に対して $CS_a(vr) \cap CS_a(vr_r) = \emptyset$ の場合のときをいう．変数 vr をレジスタ r に割当て可能なときのみ vr を r に割り当てることができ，vr を r に割り当てたとき vr は $CS_a(vr)$ において r を占有する☆．

2.3.3 データ転送路割当て

2 入力演算器 fu に演算 op が割り当てられ，レジスタ r に op の入力変数 vr が割り当てられたとき，r から fu の入力端子にデータ転送要求が発生する．このとき，レジスタから出力されるデータを fu が持つ 2 つの入力端子のうちどちらの入力端子に入力するかを決定することを端子割当て☆☆と呼ぶ．端子割当てをすることで各演算器，レジスタの接続を決定することができ，データ転送路割当てにより各演算器，レジスタ間に接続線を接続しマルチプレクサを挿入する．

2.4 接続コスト

入力端子と出力端子を結ぶ接続線を 1 本としたとき，データ転送路として使用される接続線の総和 W とマルチプレクサ数 M を用いて接続コスト C を

$$C = W + M \tag{1}$$

と表す．ここで演算器数，レジスタ数が制約として与えられ，しかもそれらが少なくとも 1 回は使用されるとき，すなわち使用されない演算器およびレジスタがないときを考える☆☆☆．すべての演算器，レジスタの入力端子の集合を P とすればその総数 $|P|$ は一定である．このとき入力端子 $p_i \in P$ に接続される接続線数を w_i とすると，2 入力 1 出力マルチプレクサ数は $w_i - 1$ で表される．これから接続線数 W とマルチプレクサ数 M は次式 (2)，(3) で表される．

$$W = \sum_{p_i \in P} w_i \tag{2}$$

$$M = \sum_{p_i \in P} (w_i - 1) = W - |P| \tag{3}$$

よって入力端子数が一定のときにはマルチプレクサ数は接続線数に比例する[15]．したがって，演算器数，レジスタ数が制約として与えられ入力端子数が一定となるときには接続コストとして W を用いることができる．以後本論文では接続コストとは接続線数の総和 W を指すものとする．

2.5 問題の定式化

以上の準備のもと，本論文で扱うリソースバインディング問題を定義する．リソースバインディング問題とは入力として，

(1) $\sigma(v)$ $(v \in V)$ が決定した DFG $G(V, E)$

(2) 演算器集合 FU

(3) レジスタ集合 R

が与えられたとき，演算器数およびレジスタ数を制約として，アーキテクチャモデルにおける接続コスト W を最小化するように，演算器割当て，レジスタ割当て，データ転送路割当てを決定することである．

3. リソースバインディングアルゴリズム

2.5 節で定義したリソースバインディング問題を解決するのにあたり，本論文では分枝限定法を採用する．これは次の 3 つの利点がある．

(1) ディジタル信号処理ハードウェアのデータパス設計を対象とした高位合成システム[12]では，リソースバインディング問題の入力となる DFG に対してすでに高速なスケジューリング，高速なリソースバインディングが実行され準最適解が得られている．この準最適解の接続コストを分枝限定法で解く際の暫定値として採用することで，準最適解よりも劣悪な解を含む解空間を探索する手間が省け効率的な解空間の探索が可能となる．

(2) 従来のリソースバインディング手法では，リソースバインディング問題を整数線形計画問題に変換しこれを解くことで，リソースバインディングを実現していた[5),10]．提案手法では，分枝限定法において分枝操作を直接演算器割当てやレジスタ割当てで実現し，限定操作において接続コストの下界値をすでに割り当てられた接続線の数え上げにより算出することで，分枝操作と限定操作を単純かつ高速にすることを可能とする．

(3) 分枝限定法は与えられた問題に対して，問題の規模が小さい場合には最適解を現実的な時間で与えることが知られている．提案手法では問題の規模が大きい場合に，分枝限定法を適用する前に発見的算法に基づく演算器割当てやレジスタ割当てによって解空間を削減し問題の規模を小さくすることで，現実的な時間で最適解に近い準最適解を求めることができる．

☆ 本論文ではレジスタのビット幅を一定と仮定するが，提案手法は複数のビット幅を持つレジスタに対しても変数を割当て可能なレジスタのビット幅に対して変数のビット幅が十分であるかを調べることにより容易に対応できる．

☆☆ 演算器 fu に割り当てられる演算 op が可換演算の場合にのみ端子割当ては実行され，非可換演算の場合には端子割当ては一意に決定される．

☆☆☆ 提案するリソースバインダは，我々が構築している高位合成システム[12]の一部として開発した．本高位合成システムではスケジューラが DFG を実行するのに必要な最小の演算器数とレジスタ数を与え，これがリソースバインダの入力となる．そのためすべての演算器とレジスタは少なくとも 1 回は使用されることになる．

3.1 アルゴリズム

リソースバインディング問題（以下原問題と記述する）に直接，分枝限定法を適用する前に，制約条件から原問題の最適解の接続コストに影響を与えない演算器割当てとレジスタ割当てをする（この2つの割当てを初期割当てと呼ぶことにする）．次に初期割当てにより割当てが確定した演算，変数を原問題から除いた部分問題に分枝操作を加えることにより，複数の部分問題を生成する．各部分問題に対して限定操作を加えることで原問題が解かれる．分枝操作は次のように実現される．与えられた未割当ての演算集合と変数集合の中から1つの演算または変数を選び，それを割当て可能な演算器またはレジスタに仮に割り当てることで解空間を削減した部分問題を生成する．限定操作は次のように実現される．得られた部分問題に対してデータ転送路割当てをし，接続コストの下界値を求める．この下界値と現段階で得られている最良の接続コスト（暫定値）を比較して，注目している部分問題が原問題の最適解を与えないことが判明すれば，その部分問題は分枝操作をされずに終端される．逆に判明しなければさらに分枝操作を加える．ある時点で部分問題の接続コストの下界値を求める際に接続コスト自身が求まれば，暫定値と比較し良ければ暫定値を更新してその部分問題を終端する．すべての部分問題が終端された時点で得られている最良解が原問題の最適解である．提案手法のアルゴリズムを以下に示す．Step 2〜7は再帰的に繰り返される．

Step 1. 初期割当て．

Step 2. 未割当て演算・変数集合から1つの未割当て演算あるいは変数を選択．

Step 3. 選択された演算あるいは変数に対して割当て可能な演算器あるいはレジスタの優先順位を決定し，優先順位の最も高い演算器あるいはレジスタを選択．

Step 4. Step 2で選択された演算あるいは変数をStep 3で選択した演算器あるいはレジスタに割り当てる．

Step 5. 端子割当て，データ転送路割当て．

Step 6. 接続コストの下界値算出．このとき未割当て演算・変数がなければ，下界値は接続コストである．

Step 7. Step 6で接続コストが得られ，接続コスト ＜ 暫定値 ならば暫定値を更新し，現時点の演算器割当て，レジスタ割当て，データ転送路割当てを暫定解として保存する．下界値 ＜ 暫定値 ならば Step 2 で選択した演算あるいは変数をStep 4で割り当てた演算器あるいはレジスタに割り当てたまま次の未割当て演算あるいは変数を選択するためにStep 2へ．下界値 ≥ 暫定値 ならばStep 3で選択した演算器あるいはレジスタをStep 2で選択した演算あるいは変数を割当て可能な演算器・レジスタ集合から除きStep 3へ．

Step 1〜6の各処理について以下で説明する．

3.2 初期割当て（Step 1）

演算 op の割当て可能演算器がただ1つのときには，演算 op をその演算器に割り当てる．同様に変数 vr を割当て可能なレジスタが1つの場合には変数 vr をそのレジスタに割り当てる．このような割当てにより原問題の解の接続コストには影響を与えない．初期の段階で，原問題の解の接続コストに影響を与えない演算器割当て，レジスタ割当てにより探索すべき解空間を削減することが初期割当ての目的である．演算は演算器に，変数はレジスタに割り当てられるので，演算と変数の場合に分けて初期割当てについて説明する．

3.2.1 演算の初期割当て

演算タイプ ot の未割当て演算集合，演算器集合を OP^{ot}，FU^{ot} とし，OP^{ot} の中でコントロールステップ $CS(i)$ $(1 \leq i \leq II)$ において実行される未割当て演算集合を OP_i^{ot} とする．このとき演算の初期割当てとは，$\max_{1 \leq i \leq II} |OP_i^{ot}|$ となるコントロールステップ $CS(i)$ において未割当て演算集合 OP_i^{ot} を割当て可能演算器に各演算タイプにつき1度のみ割当てを決定することである．こうすることで，各演算器について原問題の解の接続コストに影響を与えずに未割当て演算を各演算器に1つ割り当てることができる．

図5 は演算の初期割当ての例である．いま演算タイプ M の未割当て演算集合 $OP^M = \{op_1, \cdots, op_4\}$ と演算器集合 $FU^M = \{M1, M2\}$ が与えられたとする（図5(a)）．このとき $CS(1)$ において，$\max_{1 \leq i \leq II} |OP_i^M| = |OP_1^M|$ であるから演算 op_1，

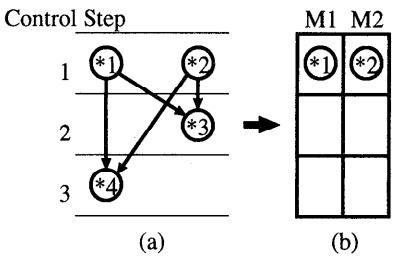


図5 演算の初期割当て．(a) スケジューリング後のDFG．(b) 演算を演算器に割当て

Fig. 5 Initial assignment of operations. (a) Scheduled DFG. (b) Operations *1 and *2 are assigned to functional units $M1$ and $M2$, respectively.

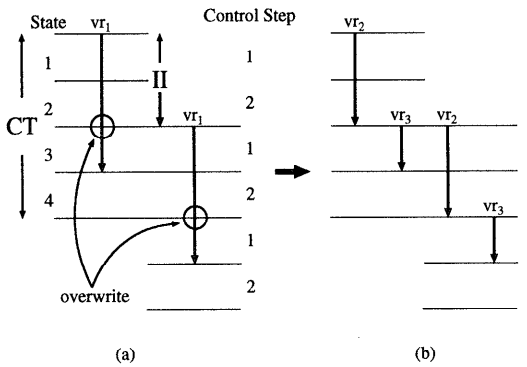


図6 変数分割. (a) 変数分割前. (b) 変数分割後
Fig. 6 Variable partitioning. (a) Before variable partitioning. (b) After variable partitioning.

op_2 を演算器 $M1$, $M2$ にそれぞれ割当てを決定することができる（図5(b)）.

3.2.2 変数の初期割当て

変数の初期割当ての前処理として入力間隔 II よりも長い生存区間長を持つ変数を2つ以上の変数に分割する. **図6** は変数分割の例である. 図6(a)では変数 vr_1 は生存区間長3の変数であり, 2イタレーション分の生存区間を示している. このとき変数 vr_1 の生存区間長は II よりも大きく, 状態1と状態3は同時に実行されるので図の円の時点で変数 vr_1 が保存されるレジスタにおいてデータの上書きが発生する. この上書きを回避するために変数の分割とレジスタ-レジスタ間データ転送が必要になる. そこで生存区間長が II よりも大きな vr_1 のような変数は II よりも短い生存区間長を持つ変数 vr_2, vr_3 に分割し, vr_3 を vr_2 と異なるレジスタに保存し, そのレジスタ-レジスタ間にデータ転送路を用意することによって変数の分割を実現する. このとき変数の分割箇所は複数通り考えられるが, 簡単のため生存区間長 II ごとに変数を分割する.

変数分割の後に変数の初期割当てをする. 変数の初期割当ては, (i) 各レジスタに変数がまだ割り当てられていないことを利用した初期割当て, (ii) 変数を割当て可能なレジスタが1つのみであることを利用した初期割当てがある.

(i) 3.2.1 項の議論と同様に $CS(i)$ $(1 \leq i \leq II)$ において生存区間の存在する未割当て変数の集合を VR_i とし与えられたレジスタ集合を R とする. このとき $|VR_i|$ が最大となる未割当て変数集合の各変数を, それぞれ1つのレジスタに割当てを決定する.

(ii) $CS(i)$ において変数に占有されていないレジスタ集合を R_i, $CS(i)$ において生存区間の存在する未割当て変数集合を VR_i とする. このとき $|R_i| = |VR_i|$ かつ, VR_i の中でレジスタ $r_a \in R_i$ に割当て可能な変数がただ1つのとき, その変数はレジスタ r_a に割当てを決定する.

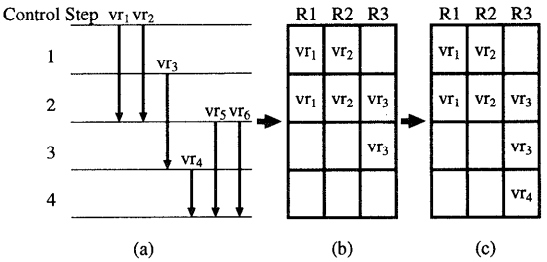


図7 変数の初期割当て. (a) 変数の生存区間表. (b) 変数の初期割当て (i). (c) 変数の初期割当て (ii)
Fig. 7 Initial assignment of variables. (a) Lifetime table of variables. (b) Initial assignment of variables (i). (c) Initial assignment of variables (ii).

図7 は変数の初期割当ての例である. いま未割当て変数 $vr_1, \cdots, vr_6$, レジスタ集合 $R = \{R1, R2, R3\}$ が与えられたとする（図7(a)）. このとき $CS(2)$ において $|VR_2| = 3$ が最大となり前述の (i) より未割当て変数 vr_1, vr_2, vr_3 をレジスタ $R1$, $R2$, $R3$ にそれぞれ割当てを決定する（図7(b)）. さらに $CS(4)$ において $|R_4| = |VR_4| = 3$ かつ VR_4 の中で $R3$ に割当て可能な変数は vr_4 のみであるので (ii) より変数 vr_4 はレジスタ $R3$ に割当てを決定することができる（図7(c)）.

3.3 未割当て演算・変数選択方法（Step 2）

未割当て演算・変数選択では, 未割当ての演算, 変数集合 OP, VR から, 演算器・レジスタに割り当てることにより接続コスト見積り値が大きくなる演算または変数を1つ選ぶ. これは接続コスト見積り値が大きくなる演算または変数を選択することで Step 7 において, 暫定値よりも接続コストの下界値が大きくなり部分問題を早い段階で終端できる可能性があるからである.

接続コスト見積り値を大きくするためには, 演算・変数を割当て可能な演算器・レジスタに割り当てた際に, 接続線数が増加する演算・変数を選択すればよい.

演算の場合, 演算 $op \in OP$ を割当て可能な演算器 fu に対して, 演算を割り当てることによって実際に増加する接続線数 $H_{op}(fu)$ を求める. op を2入力演算とすると, op を fu に割り当てることで op の出力変数 vr_o が割り当てられているレジスタ r_o へのデータ転送と, op の入力変数 vr_l, vr_r が割り当てられているレジスタ r_l, r_r から fu にデータ転送が必要となる. fu と r_o, r_l, r_r に接続線がすでに接続されて

いるかどうかを調べればよいことになり $H_{op}(fu)$ は次式 (4) で求まる．

$$H_{op}(fu) = W_{op_l}(fu) + W_{op_r}(fu) + W_{op_o}(fu) \tag{4}$$

ただし

$$W_{op_l}(fu) = \begin{cases} 1 & r_l \text{ から } fu \text{ に接続線なし} \\ 0 & \text{すでに } r_l \text{ から } fu \text{ に接続線あり} \end{cases} \tag{5}$$

$$W_{op_r}(fu) = \begin{cases} 1 & r_r \text{ から } fu \text{ に接続線なし} \\ 0 & \text{すでに } r_r \text{ から } fu \text{ に接続線あり} \end{cases} \tag{6}$$

$$W_{op_o}(fu) = \begin{cases} 1 & fu \text{ から } r_o \text{ に接続線なし} \\ 0 & \text{すでに } fu \text{ から } r_o \text{ に接続線あり} \end{cases} \tag{7}$$

op を割当て可能な演算器すべてについてこの $H_{op}(fu)$ を算出する．さらに op を fu に割り当てることによって明らかに割当てが決定する演算（3.5 節で説明する）によって増加する接続線も見積もる．つまり op を fu に割り当てることによって決定する演算 op_s の演算器 fu_s への割当てによる接続線の増加分 $H_{op_s}(fu_s)$ を $H_{op}(fu)$ に加えることで増加する接続線の見積り値 $H'_{op}(fu)$ を算出し，op の割当て可能な演算器すべてに対してこの見積り値を算出する．そのなかで最大のものを演算 op の未割当て演算（変数）を選択するうえでの評価値 H とする（式 (8)，(9)）．ただし FU_{op} は op が割当て可能な演算器集合とする．

$$H'_{op}(fu) = H_{op}(fu) + H_{op_s}(fu_s) \tag{8}$$

$$H = \max_{fu \in FU_{op}} H'_{op}(fu) \tag{9}$$

変数の場合も演算と同様にして増加する接続線の見積り値 $H'_{vr}(r)$ を算出できる．変数 $vr \in VR$ が割当て可能なレジスタ $r \in R_{vr}$ に対して，変数を割り当てることによって実際に増加する接続線数 $H_{vr}(r)$ を求める．変数 vr をレジスタ r に割り当てると，変数 vr を出力する演算 op_o が割り当てられている演算器 fu_o からのデータ転送と，変数 vr を入力とする複数個の演算 op_{in_i} が割り当てられている複数の演算器 fu_{in_i} にデータ転送が必要となる．よって $H_{vr}(r)$ は次式 (10) で求まる．

$$H_{vr}(r) = W_{vr_o}(r) + \sum_k W_{vr_{in_k}}(r) \tag{10}$$

ただし

$$W_{vr_o}(r) = \begin{cases} 1 & fu_o \text{ から } r \text{ に接続線なし} \\ 0 & fu_o \text{ から } r \text{ に接続線あり} \end{cases} \tag{11}$$

$$W_{vr_{in_i}}(r) = \begin{cases} 1 & r \text{ から } fu_{in_i} \text{ に接続線なし} \\ 0 & r \text{ から } fu_{in_i} \text{ に接続線あり} \end{cases} \tag{12}$$

この $H_{vr}(r)$ を割当て可能なレジスタごとに算出する．さらに演算の場合と同様に変数をレジスタに割り当てることで割当てが決定する（3.5 節で後述）変数 vr_s（レジスタ r_s に割当て）の接続線数増加分を $H_{vr}(r)$ に加えて見積り値 $H'_{vr}(r)$ とし，割当て可能レジスタの中で最大のものを変数 vr の未割当て演算（変数）を選択するうえでの評価値 H とする（式 (13)，(14)）．

$$H'_{vr}(r) = H_{vr}(r) + \sum_s H_{vr_s}(r_s) \tag{13}$$

$$H = \max_{r \in R_{vr}} H'_{vr}(r) \tag{14}$$

すべての未割当て演算，変数の評価値 H（式 (9)，(14)）を比較して最大のものを分枝操作によって選択される演算（変数）とする．

3.4 割当て可能演算器・レジスタの優先順位決定（Step 3）

Step 2 の未割当て演算・変数選択方法で使用した見積り値 $H'_{op}(fu)$，$H'_{vr}(r)$ を用いて，選択された演算あるいは変数が割当て可能な演算器あるいはレジスタの優先順位を決定する．優先順位は選択された演算または変数の見積り値 $H'_{op}(fu)$，$H'_{vr}(r)$ が大きいものを高く設定する．

3.5 演算–演算器・変数–レジスタ割当て（Step 4）

Step 4 では Step 2 で選択された演算・変数を Step 3 で決定された演算器・レジスタに実際に割り当てる．演算・変数を演算器・レジスタに割り当てることによって未割当ての演算・変数についても割当てを決定することができる場合がある．

選択された演算 op を演算器に割り当てた場合には，選択された演算 op の実行されるコントロールステップにおいて op と演算タイプの等しい未割当て演算を割当て可能な演算器が 1 つのみの場合には，その未割当て演算を割当て可能演算器に割当てを決定することができる．図 8 において演算 op_1 が選択され演算器 $M1$ に割り当てた場合には，演算 op_1 と演算タイプ

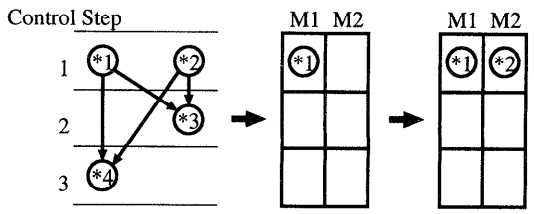


図 8 演算*1 を演算器 $M1$ に割り当てることで決定する未割当て演算*2 の 演算器 $M2$ への割当て

Fig. 8 If the operation *1 is assigned to the functional unit $M1$, the operation *2 must be assigned to the functional unit $M2$.

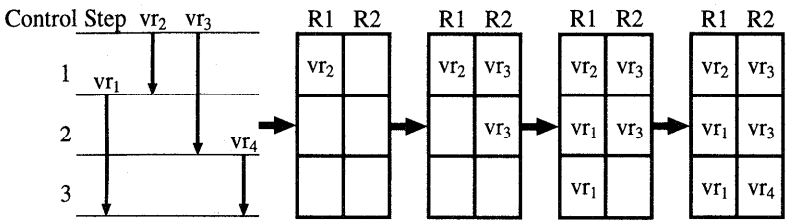


図9 変数 vr_2 をレジスタ $R1$ に割り当てることで決定する未割当て変数 vr_1, vr_3 のレジスタ $R1$, $R2$ への割当て

Fig. 9 If the variable vr_2 is assigned to the register $R1$, the variables vr_1 and vr_3 must be assigned to the registers $R1$ and $R2$, respectively.

が等しく同じコントロールステップで実行される未割当て演算 op_2 を演算器 $M2$ に割当てを決定することができる.

選択された変数をレジスタに割り当てた場合には, 選択された変数と生存区間に重複のある未割当て変数を割当て可能なレジスタが1つのみの場合, その未割当て変数の割当てを決定することができる. さらに前述の 3.2.2 項 (ii) の場合にも割当てを決定することができる. **図9** において変数 vr_2 が選択されレジスタ $R1$ に割り当てた場合には, 変数 vr_2 と生存区間に重複のある未割当て変数 vr_3 を割当て可能なレジスタは $R2$ のみなので変数 vr_3 のレジスタ $R2$ への割当てが決定する. また変数 vr_3 をレジスタ $R2$ に割り当てることで vr_3 の生存区間と重複のある未割当て変数 vr_1 がレジスタ $R1$ に割当てが決定し, 同様にして変数 vr_4 はレジスタ $R2$ に割当てが決定する.

3.6 端子割当て・データ転送路割当て (Step 5)

端子割当てとはレジスタから出力されるデータを演算器のどちらの入力端子に入力するかを決定する処理であり, データ転送路割当てとは端子割当てを基に接続線を接続しマルチプレクサを挿入することである.

演算器 fu に割り当てられたすべての2入力可換演算☆について, 対となる入力変数が割り当てられた2つのレジスタから演算器 fu の同一入力端子に接続線を接続しないようにすべての接続を試す. 接続線数が最小となる接続が求める端子割当てである.

端子割当ての次に, データ転送路割当てをする. データ転送路割当ては接続要求のある演算器–レジスタ間に接続線を接続する. 1入力端子に2本以上の接続線が入力される場合にはマルチプレクサを挿入する.

3.7 接続コスト下界値算出方法 (Step 6)

接続コストの下界値を算出する. データ転送要求のある演算, 変数間で演算器割当てとレジスタ割当てが済んでいると, 端子割当てやデータ転送路割当てにより接続線が接続され, 既存の接続線数を算出することができる. しかし, データ転送要求のある演算, 変数間で演算器割当てとレジスタ割当ての両方が済んでいなくとも, 割当てが進めば必ず接続線が増加することがこの時点で判別できる場合がある. その接続線数を既存の接続線数に加えることで接続コストの下界値とする.

演算 op から変数 vr にデータ転送要求が存在し, op は演算器 fu に割当て済であり vr は未割当てであるとする. このとき変数 vr を割当て可能なレジスタ集合 R_{vr} のすべてのレジスタに fu から接続線が接続される可能性がある. 仮に R_{vr} のどのレジスタにも fu から接続線がないものとすると, 必ず接続線は1本必要となる. この1本を接続コストの下界値として加える. また R_{vr} の少なくとも1つのレジスタに fu から接続線が接続されていれば接続コストの下界値に変化はない. データ転送要求のある演算, 変数間で演算器割当てまたはレジスタ割当てのどちらかが済んでいるすべての場合について以上の処理をすることで接続コストの下界値が算出できる.

3.8 発見的算法に基づく初期割当て

与えられた問題の規模が大きい場合や初期割当ての際に解空間の削減が効率良くされなかった場合には, 分枝限定法が探索する解空間を現実時間内では探索が不可能になる. このとき, 初期割当て (Step 1) の後に発見的算法に基づく初期割当てをすることで解空間を初期割当て後よりも小さいものにできれば, 上述のような場合にも現実的時間内で解が得られる. また発見的算法に基づく初期割当てをする際に明らかに不適当な割当てをしなければ, 削減された解空間に最適解が含まれる可能性は少なくなり最適解に近い準最適解を得ることができる.

リソースバインディング問題の目的関数は接続コス

☆ 2入力可換演算とは, 加算や乗算のように入力変数に交換法則の成り立つ演算のことである.

トの最小化であるので，接続コストが明らかに増加する割当てが不適当な割当てと考える．接続コストの増加を防ぐことは，複数のデータ転送要求を1つのデータ転送路で担うことにより実現される．このことから，複数のデータ転送を1つのデータ転送路で担うように演算，変数を演算器，レジスタに割り当てることが不適当ではない割当てと考えることができる．これを発見的算法に基づく割当てとする．

複数のデータ転送要求を1つの接続線で担う割当てには次のような場合が考えられる．

(A) 同一の変数を入力とする演算タイプが等しい複数の演算を1つの演算器に割り当てる場合．

(B) 同一の演算器に割り当てられている演算が出力する変数を1つのレジスタに割り当てる場合．

ここでは(A)の場合について説明する．変数 vr を入力とする演算タイプが等しい演算集合を OP とする．このとき OP にすでに演算器 fu に割り当てられた演算 op が含まれる場合には，fu に割当て可能な OP 内の演算をすべて fu に割り当てる．Bの場合もAと同様の考えで割り当てることができる．

前述した3.2節の初期割当ては，最適解を含む解空間を残すことを保証して解空間を削減し後段の分枝限定法で探索する解空間を減らすことが目的であるが，発見的手法に基づく初期割当ては最適解を含む解空間を残すことを保証せずに，3.2節の初期割当てに比較して分枝限定法で探索する解空間を大きく減らすことを目的としている．よって3.2節の初期割当ておよび発見的手法に基づく初期割当てによって演算および変数が演算器およびレジスタに割り当てられた場合，この割当ては再び変更されることはない．

4. 計算機実験結果

提案手法をSUN Ultra Sparc上にC言語で実装し次の実験をした．

実験1 文献10)の手法（ILP），シミュレーテッド・アニーリング法（SA）と本手法（Ours1）の比較．文献10)の方法でリソースバインディング問題を整数線形計画問題に変換しILPパッケージ[16]で解いたもの，およびリソースバインディング問題をSA法で解いたものと本手法との実行時間を比較する．ただしSA法の実験条件は初期確率が0.99，温度降下係数0.95，終了温度0.5および各温度での繰返し回数は演算と変数の総和の100倍である．入力として与えるDFGは3次IIRフィルタ（IIR3）と5次IIRフィルタ（IIR5）である．

実験2 5th order Elliptic Wave Filter（EWF）のDFG（節点数36個）に対する各手法LYRA[2]，ARYL[2]，SPLICER[7]，HAL[9]，BINET[11]，STAR[13]，本手法（Ours1）との実行時間と接続線数，マルチプレクサ数の比較．

実験3 本手法（Ours1）と3.8節で述べた発見的算法により初期割当てをした場合の本手法（Ours2）の解空間の比較．入力として与えたDFGは実験1のIIR3，IIR5，実験2のEWFと40次FIRフィルタ（FIR40）（節点数85個），50次FIRフィルタ（FIR50）（節点数103個）である．

実験1，2，3の結果を**表1**，**表2**および**表3**に示す．表中の#nodesはDFGの節点数，CTはコンピュテーションタイム，IIは入力間隔，#(adds, muls, regs)は加算器数，乗算器数，レジスタ数，#wiresは接続線数，#MUXは2入力1出力換算のマルチプレクサ数，Timeは実行時間，#PPは分枝操作によって得られる部分問題の総数，RatioはOurs1とOurs2の部分問題の総数の比を表す．本手法に与えるDFGのスケジューリング結果は高位合成システム[12]の最適解を保証するスケジューラの出力を用いた．接続線数の暫定値はすべて100とした．

実験1から，リソースバインディング問題を整数線形計画問題に変換することによる解法よりも提案手法が高速に最適解を算出できることを示している．さらに，シミュレーテッド・アニーリング法に比較して提案手法は高速に最適解を算出できることを示している．これらの結果から，リソースバインディング問題に対し分枝限定法における分枝操作・限定操作を3.4〜3.7節のように実現することで演算器割当て，レジスタ割当て，データ転送路割当てを分割することなく従来手法に比較して高速に最適解を算出することができ，提案手法の有効性を確認できた．

実験2の結果からEWFに対し，本手法（Ours1）は既存手法[2),7),9),11),13)]と比較して接続線数，およびマルチプレクサ数の両面で少ない解を得ている．

実験3の結果から発見的算法による初期割当てをすることで，発見的算法による初期割当てをしなかった場合に比べ解空間を最大99%，平均40%削減したことを確認した．計算時間についても同様の結果が得られている．また発見的算法による初期割当てをすることで配線本数は増加しているが10%程度の増加に抑えられている．なお表3のFIR50において，Ours1では127時間にわたり解探索した結果最適解を得たのに対し，Ours2では4.9秒で，接続線数108という最適解と等しい解を得た．このことから，提案手法は比較的大規模のリソースバインディング問題に対しても，

表1 SA, ILPと本手法の比較
Table 1 Comparison of SA, ILP and ours.

Input Data					ILP		SA		Ours1	
DFG	#nodes	CT	II	#(adds,muls,regs)	#wires	Time	#wires	Time	#wires	Time
IIR3	16	10	5	(2, 2, 7)	22	2.5 min	24	5 min	22	2 sec
	16	10	6	(1, 2, 6)	20	2.3 sec	22	6 min	20	0.1 sec
	16	10	7	(2, 2, 7)	22	1 min	24	5 min	22	0.3 sec
	16	11	6	(1, 2, 6)	20	2.4 sec	22	6 min	20	0.1 sec
IIR5	25	14	9	(2, 2, 9)	32	11.5 hours	36	35 min	32	32 min
	25	14	12	(1, 2, 9)	28	9 min	31	48 min	28	4.5 min
	25	15	8	(2, 2, 9)	32	4 hours	35	30.5 min	32	1 min
	25	16	9	(2, 2, 8)	29	41 hours	31	32 min	29	17 min

表2 EWFの合成結果
Table 2 Synthesis results for EWF.

System	CT	II	#(adds,muls,regs)	#MUXs	#wires	Time
LYRA [2]	21	21	(2, 1, 10)	23	47	3 sec
ARYL [2]	21	21	(2, 1, 10)	22	46	1 sec
SPLICER [7]	21	21	(2, 1, 24)	14	43	> 50 sec
HAL [9]	19	19	(2, 1, 12)	20	45	6 min
BINET [11]	17	17	(2, 1, 10)	N.A.	40	9 sec
STAR [13]	17	17	(2, 1, 10)	21	43	10 sec
Ours1	21	21	(2, 1, 10)	20	36	6 hours
	19	19	(2, 1, 10)	21	37	3.5 hours
	17	17	(2, 1, 10)	21	37	87.5 hours

表3 発見的算法により初期割当をした場合の解空間の削減率
Table 3 Comparison of solution space.

Input Data			Ours1			Ours2			
DFG	CT	II	#PP†	#wires	Time	#PP†	#wires	Time	Ratio‡
IIR3	10	5	477	22	0.9 sec	346	22	0.7 sec	72.5%
	10	6	51	20	0.1 sec	18	20	0.1 sec	35.3%
	10	7	55	22	0.1 sec	17	23	0.1 sec	30.9%
	11	6	51	20	0.1 sec	18	20	0.1 sec	35.3%
IIR5	14	9	200803	32	32 min	34595	32	1.5 min	17.2%
	14	12	7486	28	30.5 sec	167	28	1 sec	2.23%
	15	8	8371	32	35.0	6104	33	25 sec	72.9%
	16	9	45757	29	4 min	5151	29	24 sec	11.3%
EWF	21	21	1272459	36	6 hours	95163	37	32 min	7.5%
	19	19	65805	37	3.5 hours	8334	40	9.5 min	1.21%
	17	17	10766738	37	87.5 hours	56010	38	44 min	0.52%
FIR40	44	44	79	86	19.1 sec	1	86	0.6 sec	1.26%
FIR50	53	53	39325881	108	127 hours	100	108	4.9 sec	0.001%

† 分枝操作によって得られる部分問題の総数.

‡ Ours1 と Ours2 の部分問題の総数の比.

現実的な時間で最適解にきわめて近い準最適解を得られることが期待できる.

このことから最適性を犠牲にしても発見的算法による初期割当てをする有効性を確認できた.

5. おわりに

本論文では,ディジタル信号処理ハードウェアのデータパス設計を対象としたリソースバインディング手法を提案した.提案手法は,分枝限定法をリソースバインディング問題に直接適用することで,リソースバインディング問題を演算器割当て,レジスタ割当て,データ転送路割当てに分割することなく,1つの問題として扱うことを可能とした.計算機実験の結果,提案手法は従来のリソースバインディング手法と比較して高速に最適解を得られることを確認した.

現在構築している高位合成システム[12]は,図2のようにマルチプレクサのみを転送路として用いているハードウェアアーキテクチャをモデルとしているが,今後バス構造を持ったハードウェアアーキテクチャを導入する予定である.これにともない提案手法も今後の課題としてデータ転送路としてマルチプレクサだけでなく,バスを考慮できるよう拡張していくことを考えている.

謝辞 本研究の一部は,文部省科学研究費補助金(基盤研究C(2),課題番号10650345および奨励研究(A),課題番号10750303)の援助を受けた.

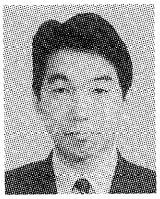
参考文献

1) Devadas, S. and Newton, A.R.: Algorithm for hardware allocation in data path synthesis, *IEEE Trans. Computer-Aided Design*, Vol.8, No.7, pp.768–781 (1989).
2) Huang, C.-Y., Chen, Y.-S., Lin, Y.-L. and Hsu, Y.-C.: Data path allocation based on bipartite weighted matching, *Proc. 27th Design Automation Conf.*, pp.499–504 (1990).
3) 茨木俊秀:組合せ最適化の理論, pp.137–172, 電子通信学会 (1980).
4) 加藤健吉,戸川 望,佐藤政生,大附辰夫:接続コストの最小化を目的とした高速アロケーション手法,電子情報通信学会信学技報, VLD-96-96, pp.1–8 (1997).
5) Liu, T.Y. and Lin, Y.L.: FLORA: A data path allocator based on branch-and-bound search, *INTEGRATION*, pp.43–66 (1991).
6) Ly, T.A. and Mowchenko, J.T.: Applying Simulated Evolution to High Level Synthesis, *IEEE Trans. Computer-Aided Design*, Vol.12 pp.389–409 (1993).
7) Pangrle, B.M.: Splicer: A heuristic approach to connectivity binding, *Proc. 25th Design Automation Conf.*, pp.536–541 (1988).
8) Parker, A.C., Mlinar, M. and Pizarro, J.: MAHA: A program for data path synthesis, *Proc. 23rd Design Automation Conf.*, pp.461–466 (1985).
9) Paulin, P.G. and Knight, J.P.: Scheduling and binding algorithm for high level synthesis, *Proc. 26th Design Automation Conf.*, pp.1–6 (1989).
10) Rim, M., Jain, R. and Leone, R.D.: Optimal allocation and binding in high-level synthesis, *Proc. 29th Design Automation Conf.*, pp.120–123 (1992).
11) Rim, M., Mujumdar, A., Jain, R. and Leone, R.D.: Optimal and heuristic algorithms for solving binding problem, *IEEE Trans. VLSI Systems*, Vol.2, No.2, pp.211–225 (1994).
12) Togawa, N., Hisaki, T., Yanagisawa, M. and Ohtsuki, T.: A high-level synthesis for digital signal processing based on enumerating data-flow graphs, *Proc. ASP-DAC '98*, pp.265–274 (1998).
13) Tsai, F.S. and Hsu, Y.C.: STAR: An automatic data path allocator, *IEEE Trans. Computer-Aided Design*, Vol.11 pp.1053–1064 (1992).
14) Tseng, C.-J. and Siewiorek, D.P.: Automated synthesis of data paths in digital systems, *IEEE Trans. Computer-Aided Design*, Vol.5, No.3, pp.379–395 (1986).
15) 山田正一郎:VLSIデータパスアロケーションにおけるブロック端子割当問題と一解法,電子情報通信学会第9回回路とシステム軽井沢ワークショップ論文集, pp.371–376 (1996).
16) ftp://ftp.es.ele.tue.nl/pub/lp_solve/

(平成10年9月18日受付)
(平成11年2月8日採録)

中本 真児

昭和48年生.平成9年早稲田大学理工学部電子通信学科卒業.平成11年同大大学院修士課程修了.同年松下通信工業(株)入社.電子回路の設計自動化,特に高位設計の自動化に関する研究に従事.



戸川 望（正会員）

昭和45年生．平成4年早稲田大学理工学部電子通信学科卒業．平成6年同大大学院修士課程修了．平成9年同後期課程修了．博士（工学）．現在同大学理工学部電子・情報通信学科助手．電子回路の設計自動化，計算幾何学，グラフ理論等の研究に従事．平成8年第9回安藤博記念学術奨励賞受賞．平成9年度（第21回）丹羽記念賞受賞．IEEE，電子情報通信学会各会員．

柳澤 政生（正会員）

昭和34年生．昭和56年早稲田大学理工学部電子通信学科卒業．昭和58年同大大学院博士前期課程修了．昭和61年同後期課程修了．工学博士．昭和59年同大学情報科学研究教育センター助手．昭和61年カリフォルニア大学バークレー校研究員．昭和62年拓殖大学工学部情報工学科助教授を経て，現在早稲田大学理工学部電子・情報通信学科教授．電子回路の設計自動化，ノイズ解析，計算幾何学，グラフ理論等の研究に従事．昭和62年度丹羽記念賞受賞．平成2年安藤博学術奨励賞受賞．IEEE，ACM，電子情報通信学会，プリント回路学会，日本OR学会各会員．

大附 辰夫（正会員）

昭和15年生．昭和38年早稲田大学理工学部電気通信学科卒業．昭和40年同大大学院修士課程修了．同年日本電気（株）入社．昭和55年同退社．現在，早稲田大学理工学部電子・情報通信学科教授．工学博士．電子回路のCADおよびこれに関連した基礎研究に従事．昭和44年度電子情報通信学会論文賞受賞．1974年IEEE CAS Societyより Cuillmin-Cauer 賞受賞．1995年IEEE CAS Societyより Meritorious Service Award 賞受賞．1994年第32回電子情報通信学会業績賞受賞．共著「VLSIの設計I」（岩波書店），編共著「Layout Design and Verification」（North-Holland）．電子情報通信学会，電気学会，プリント回路学会各会員．IEEE Fellow．