DA－90－27

DSP向けリターゲッタブルコンパイラの演算器/転送経路のバインディング手法

服部 靖史† 石浦 菜岐佐† 山口 雅之†,††

†大阪大学大学院工学研究科情報システム工学専攻
〒565-0871 大阪府吹田市山田丘 2-1
Phone: 06-879-7808, Fax: 06-875-5902
E-mail: {dankichi, ishiura, yamaguti}@ise.eng.osaka-u.ac.jp

††シャープ株式会社 IC 事業本部 設計技術開発センター
〒632-8567 奈良県天理市櫟本町 2613-1
Phone: 0743-65-2531, Fax: 0743-65-3814
E-mail: masa@edag.ptdg.sharp.co.jp

あらまし

本稿では, DSP 向けリターゲッタブルコンパイラにおける演算/転送のバインディングの新しいアルゴリズムを提案する. リターゲッタブルコンパイラのバインディング手法としてこれまでに, 演算のバインディングを BDD を用いて行なった後に転送経路のバインディングをバックトラックにより求めるという手法を提案してきたが, アーキテクチャや入力プログラムによっては非常に多くのバックトラックが発生するという問題があった. これを解決するため, 本稿では演算のバインディングと転送のバインディングをまとめて 1 つの問題として定式化し, BDD または整数線形計画法によりこれを解く方法を提案する.

キーワード リターゲッタブルコンパイラ, バインディング, デジタル信号プロセッサ

Operation and Transfer Binding of Retargetable Compilation for DSP

Yasushi HATTORI†, Nagisa ISHIURA†, and Masayuki YAMAGUCHI†,††

†Dept. Information Systems Eng,
Osaka University
2-1 Yamada-Oka, Suita, Osaka, 565-0871 Japan
Phone: +81-6-879-7808, Fax: +81-6-875-5902
E-mail: {dankichi, ishiura, yamaguti}@ise.eng.osaka-u.ac.jp

††Design Technology Development Center, IC Group,
SHARP Corporation
2613-1 Ichinomoto-cho, Tenri, Nara, 632-8567 Japan
Phone: +81-743-65-2531, Fax: +81-743-65-3814
E-mail: masa@edag.ptdg.sharp.co.jp

Abstract

This paper presents a new operation and transfer binding algorithm for a retargetable compiler for digital signal processors. In our previous method, operation binding and transfer binding were solved separately by using BDD and back-tracking, respectively. However this mehod fails to find solutions depending on architectures and programs, due to frequent back-tracking. In this paper, we propose an algorithm to solve the both binding problems at a time using BDD or Integer Linear Programming.

key words retargetable compiler, binding, digital signal processor

1 はじめに

近年の半導体技術の進歩に伴う集積度の向上に伴い, 大規模システムを 1 チップ上に集積するシステム・オン・チップの実現が可能になってきた. 複雑な機能を短期間にかつ柔軟に実現するため, 応用に特化したハードウェアを備えた CPU/DSP コアとソフトウェアの組合せによってシステムを実現することが多くなってきた. このようなシステムの設計を効率化するためには, 機能合成, 論理合成等のハードウェアの設計支援とともに, CPU/DSP コアに対するコードを生成するためのコンパイラの技術が重要になっている.

システム LSI の DSP コア設計で用いるコンパイラには, (1) 複雑なデータパスアーキテクチャに対するコードを生成できること, および (2) リターゲッタビリティを有することが要求される. DSP では必要な性能を達成しつつ徹底的な低コスト化, 低消費電力化を図るために, 汎用プロセッサとは異なる複雑なアーキテクチャがとられるため (1) が必要となる. また, 応用毎に最適なデータパス構成を備えたり, 要求性能を満たすための設計変更がしばしば行なわれるプロセッサに対して個々にコンパイラを開発することは非効率であることから (2) が重要な要件となる.

リターゲッタブルコンパイラの利用により, (1) 高級言語による特定用途向きプロセッサのソフトウェア開発, (2) ハードウェアの設計と並行したソフトウェア開発, (3) ハードウェアの設計変更に対するソフトウェア再設計の省力化, などのメリットが得られるため, リターゲッタブルコンパイラに関する研究は多く行なわれている [Mar95, Mar97, Lie95, Pra96, Aka93, Sat94].

これらの主な目標としては, 生成コードの品質向上と, リターゲッタビリティ向上との 2 つがあるが, 本研究は後者に焦点を当てるものである. 従来のリターゲッタブルコンパイラでは, 汎用プロセッサの単純な拡張を対象とするものが多かったが, DSP では, 図 1 のような「非直交な」データパスをとられることが多い. 非直交なデータパスには, 汎用レジスタではなく異なる特定用途のレジスタが散在し, 任意のレジスタと演算器の間でデータを授受できるとは限らない.

例えば,「y ← (a +b)*c」という計算を図 1のデータパスで行なうことを考える. a と b の加算は加算器 ADD か ALU で行なえるが, もしこの加算を ADD に割当てる (バインディングする) と, その結果は乗算器に転送することができず (ADD の出力はレジスタ A を介して RAM のアドレス入力にしか到達できないため), 以後の計算を行なうことができない. CHESS[Pra96] では, 不規則なデータパスにおいて演算器間の転送経路をパス探索により求める方法を用いているが, 非直交なデータパスにおいて, バインディングによってはデータ転送が不可能になるという問題を完全には解決していなかった.

これを解決するため, 我々はこれまでに, データ転送経路が必ず存在するような演算の演算器への割当て (演算バインディング) をまず求め, 次に転送資源の衝突を考慮してデータ転送の経路割当て (転送バインディング) を求めるという手法を提案してきた. 演算のバインディングはバインディングのすべての可能性を列挙したバインディング空間グラフ BSG (Binding Space Graph) というグラフを作成し, BSG に DFG を埋め込む問題を BDD を用いて解くことによって求め, 転送経路のバインディングはバックトラックにより求めるというアプローチをとっていた. しかし, この方法ではアーキテクチャによっては非常に多くのバックトラックが生じ実際的な時間内に解が得られない場合が存在することが判明した. そこで本稿では, 演算バインディングと転送バインディングを同時にモデル化した拡張バインディング空間グラフ EBSG (Extended Binding Space Graph) を導入し, 両バインディングで要求される制約を全て列挙することで, 上記の問題を解決する手法を提案する.

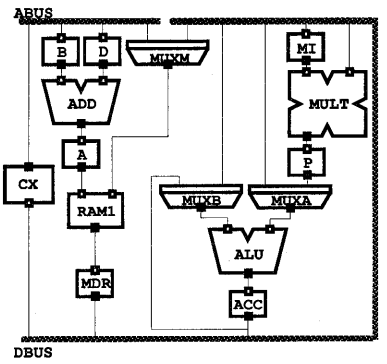


図 1: 不規則な構成を持つプロセッサ.

以下, 2 節ではバインディング問題について述べ, 3 節でバインディングアルゴリズムの記述を行なう. 4 節で実験結果と考察を示し, 5 節でまとめを行なう.

2 バインディング問題

2.1 演算バインディングと転送バインディング

バインディングとは与えられたデータフローグラフ (DFG) の演算やデータ転送をデータパス中の演算器やレジスタ, 転送経路に割当てることである. 本稿では, バインディングをさらに演算バインディングと転送バインディングに分けて考える.

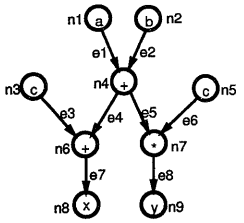


図 2: DFG.

演算バインディングとは, DFG の節点のデータパス上の演算器 (入出力節点の場合はレジスタ) への割当てを行なうことである. 図 2の DFG を図 1 のデータパスへバインディングする場合, 入力節点 a, b はデータパス中のいずれかのレジスタに, 加算は ADD, ALU のいずれかに, 乗算は MULT に割り当てられる. 任意のレジスタ及び演算器間にデータ転送経路が保証されない非直交なデータパスでは, 割当てによっては計算が不可能になることがある. 例えば変数 a, b が MDR, ACC にそれぞれ割当てられているとき, もし n4 の加算をアドレス計算用の ADD に割当てを行なうと, この結果は RAM のアドレスポートにしか収めることができず, このため乗算を行なう MULT や, 次の加算を行なう ALU あるいは ADD へのデータ転送ができなくなる. 従って, n4 は ALU へ, n6 は ALU あるいは ADD へ, n7 は MULT へ割当てなければならない.

転送バインディングは, DFG の枝の転送経路への割当を求めることである. この際に, 複数の転送経路間で資源の衝突が起こらないようにしなければならない. 図 1のデータパスにおいて, MDR と ACC に割当てられたデータを ALU で加算することを考える. MDR からも ACC からも ALU へのパスが存在しているが, ACC から ALU への転送に DBUS と MUXB を経由する経路を選択すると, DBUS の資源競合のため, MDR から ALU へのデータ転送を同時に行なうことができなくなってしまう. この場合は, ALU へのデータ転送時に DBUS を使わないようにするか, あるいは, どちらかのデータを MULT 内を経由させてレジスタ P などに格納することでこの競合を避けることが必要となる.

2.2 従来手法

これまでに我々は,

1. 演算バインディング問題を, 与えられた DFG とデータパスから導かれるバインディング空間グラフ (BSG) に, DFG を埋め込む問題として定式化し,
2. 得られた演算バインディングに対して資源の衝突がない転送経路バインディングをバックトラックにより求める.

という手法でバインディング問題を解いてきた [Yam97-1, Yam97-2, Yam98].

図 3に図 1のデータパスと図 2の DFG に対する BSG を示す. BSG の節点は部分集合 $N_1, \cdots, N_7$ にグループ化されている. ここで, 各 N_i は DFG 中の演算 n_i と一対一に対応しており, N_i は演算 n_i の可能な割当てを表す. 例えば N_4 は, 演算 n_4 が ADD, ADD', ALU, ALU' のいずれかに割当て可能であることを表している. ここで ADD', ALU' はそれぞれ ADD, ALU の入力を交換したものである. また, DFG の入力値と出力値は, MDR, ACC にのみ格納するものと仮定している. BSG の枝は節点に割当てられた演算資源間に転送経路が存在することを示している. 転送経路は複数のレジスタを経由してもよいし, ALU の「スルー演算」(一方の入力をそのまま出力に出す) を利用するものでもよい.

バインディング問題は, DFG を BSG に埋め込む問題と考えることができる. 図 3の太線部が 1 つの解である. この問題は BDD [Min90] やグラフの変形 [Yam98] により解くことができる. こうして得られた演算バインディングは少なくとも 1 組以上の転送経路を持つので, この中から資源の衝突の起きないものを探す. もし, そのような転送経路の組合せがなければ別の演算バインディングを求める. 同時に動作可能な演算器へのデータ転送が独立に行なえるように設計されたデータパスでは転送経路探索の失敗はほとんど起きない. しかし, コスト削減のためバスの本数を減らしたようなデータパスでは転送経路の資源衝突によるバックトラックが頻発し, 効率良く解を求められないという問題が生じる.

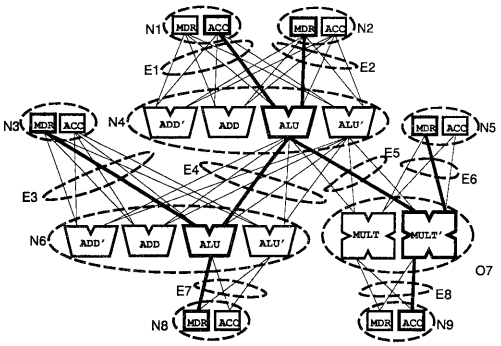
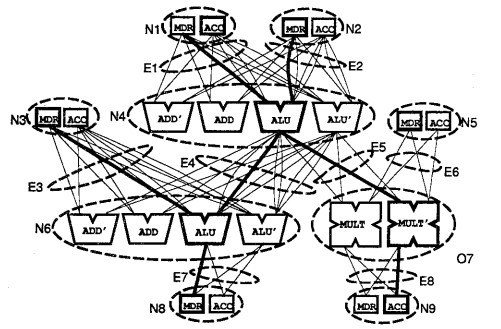


図 3: BSG.

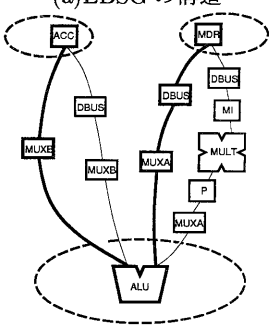
2.3 EBSG を用いた演算/転送バインディング

この問題を解消するため, 本稿では, 演算バインディングと転送経路バインディングを同時に解く方法を提案する. 演算資源間の転送経路が複数ある場合にはそれら全てを記述する EBSG (拡張 BSG) を定義し, 演算/転送経路バインディング問題を DFG の EBSG への制限付き埋め込み問題として定式化する.

図 4に図 1のデータパスと図 2の DFG に対する EBSG を示す.



(a)EBSG の構造



(b) 枝のラベル

図 4: EBSG.

EBSG の各節点は BSG のそれと同じであり, EBSG の節点グループ $N_1, \cdots, N_9$ はそれぞれ DFG の節点 n_1 から n_9 に対応するものである. EBSG の各枝は, 演算資源間の 1 つの転送経路に対応し, 転送経路上の演算資源のリストでラベル付けされている (図 4(b)). 演算器間に複数の転送経路が存在する場合には, それに対応して複数の枝が存在する. 図 4(b) は N_1 の ACC と N_4 の ALU および N_2 の MDR と N_4 の ALU の間の枝のラベルを示している. 例えば, o_1 を ACC に, o_2 を MDR に, o_4 を ALU に割当てた場合, ACC → ALU, MDR → ALU にはそれぞれ 2 通りのパスがあり, それぞれラベルで示された資源を経由してデータが転送されることを表している.

我々の目的は DFG の節点を EBSG の節点に, DFG の枝を EBSG の枝に割当て, かつ EBSG の枝にラベル付けされた転送経路の間で資源衝突が生じないようにすることである. つまり, 演算/転送バインディング問題は, DFG の BSG への制限つき埋め込み問題として定式化できる.

3 バインディングアルゴリズム

3.1 概要

DFG の BSG への埋め込み問題に対して, BDD や整数線形計画法を利用して解く手法 [Yam97-1, Yam97-2] と, グラフの変形を行なって解く手法 [Yam98] とがある. 本稿で提案する手法は, 前者を EBSG 向けに拡張したものである.

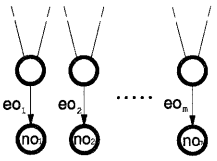
DFG の枝 e_iに対応する EBSG の枝の集合を $E_i = \{e_{i,1}, e_{i,2}, \cdots, e_{i,m}\}$ とする. EBSG 内の各枝 $e_{i,j}$ に, 0-1 変数 $x_{i,j}$ を割当てる. $x_{i,j} = 1$ は枝 $e_{i,j}$ が埋め込みに選択され, $x_{i,j} = 0$ は選択されないことを表す.

本手法では DFG の EBSG への埋め込みに必要な条件を全て列挙し, BDD や整数線形計画法を用いてこれらの条件を満たす解を求める.

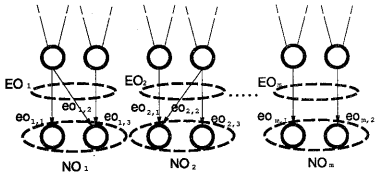
3.2 制約条件

DFG が EBSG に埋め込める条件は, 次の 4 つである.

1. 出力節点存在条件



(a)DFG



(b)EBSG

図 5: 出力節点存在条件.

DFG の出力節点を $no_1, \cdots, no_m$ とし, その節点に接続する枝を $eo_1, \cdots, eo_m$ とする (図 5(a)). また, $eo_1, \cdots, eo_m$ に対応する EBSG の枝の集合を $EO_1, \cdots, EO_m$ とする (図 5(b)). この時の $no_1, \cdots, no_m$ 出力値が計算されるためには EO_i に含まれる EBSG の枝 $eo_{i,1}, \cdots, eo_{i,n}$ の中から, 必ず 1 本の枝が選ばれなければならない. この制約は以下のように表すことができる.

$$\forall EO_i (i = 1, \cdots, m) : \sum_{e_{i,j} \in EO_i} x_{i,j} = 1$$

2. 入力データ転送条件

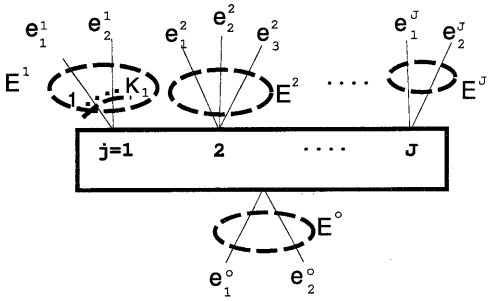


図 6: 入力データ転送条件.

J 入力の EBSG の節点において, $j(1 \leq j \leq J)$ 番目の入力ピンに接続する枝の集合を, $E^j = \{e_1^j, \cdots, e_{K_j}^j\}$ とする. また, 出力ピンに接続する枝の集合を $E^o = \{e_1^o, \cdots, e_{K_o}^o\}$ とする.

今, E^o 内の枝 $e_k^o (1 \leq k \leq K_o)$ が選択されたとする. これは, この EBSG 節点に対応する演算器の計算する値が用いられるということを意味し, そのためには各入力ピンに対して 1 本ずつ EBSG の枝が選択されなければならない. この制約は以下のように表すことができる. ただし, x_i^j は枝 e_i^j に対応する変数である.

$$\forall e_k^o (k = 1, \cdots, K_o) : x_k^0 \rightarrow \bigwedge_{1 \leq j \leq J} \left(\sum_{e_i^j \in E^j} x_i^j = 1 \right)$$

3. 枝の排他条件

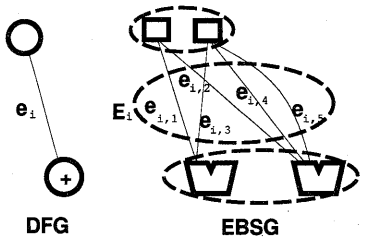


図 7: 枝の排他条件.

DFG の枝 e_i に対応する EBSG の枝の集合を E_i とする時, 枝 E_i 内の各枝 $e_{i,j}$ は 2 本以上選ばれてはならない. この制約は以下のように表すことができる.

$$\forall E_i : \sum_{e_{i,j} \in E_i} x_{i,j} \leq 1$$

4. 資源制約条件

同一のクロックサイクル内で同じ演算資源やセレクタが複数のデータ転送に用いられてはならない.

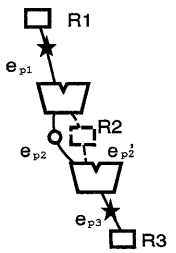


図 8: 資源制約条件.

図 8 において, R1, R2, R3 はレジスタである. また, 「★」や「○」は枝にラベル付けされている演算資源を表している. 図の実線のようなパ

スを選択した時, e_{p1} と e_{p3} に同じ資源が存在し, 資源衝突を起こすため, この 3 つの枝が同時に選択されてはならない. e_{p2} ではなく e'_{p2} が選択された時は, e_{p1} と e_{p3} の間にレジスタが存在しているため両者は同一クロックサイクル内にはなく, 共存することができる.

同じ資源が 2 つの枝 e_{p1}, e_{pm} にラベル付けされており, 両者の間にレジスタを介さないパス $\{e_{p1}, \cdots, e_{pm}\}$ が存在しているとする. この時資源制約は,

$$\sum_{i=1}^{m} x_{pi} < m$$

と書ける. この制約は, 同じ資源をラベルに含む全ての枝の対 (e_{p1}, e_{pm}) の間のレジスタを介さないパスに対して要求される.

3.3 解法

上記の制約条件を全て満たすような $x_{i,j}$の値の組合せを求めれば, DFG の EBSG への埋め込みが見つかったことになる. $x_{i,j}$の組合せは, BDD あるいは整数線形計画法により求めることができる.

BDD を用いる場合には, 制約条件を論理式の形に変換し, これらの論理積を表す BDD を構築してその値が 1 となる入力変数の組合せを求めればそれが解となる. 整数線形計画法を用いる場合には, 制約条件を一次不等式の形に変換し, これらを制約として実行可能解を求めればよい.

4 実験結果

前節に述べた手法を UNIX ワークステーション上に BDD パッケージ, 整数線形計画法のパッケージを用いて C++で実現した. BDD パッケージには [Min90] を, 整数線形計画法のパッケージには GAMS/OSL を用いた. 図 4の節点数が 22 個, 枝の数が 61 本の EBSG の例に対し, Indy MIPS R4600 (100MHz) 上で, BDD パッケージを用いた方法では 182.3 秒, 整数線形計画法を用いた方法では 2.7 秒で解を得ることができた.

5 結論

本稿では, EBSG を用いて演算バインディングと転送バインディングを同時に解く手法を提案した. これにより, バックトラッキングにより転送バインディングを求める手法が失敗した例に対しても解を求めることができると考える.

反面, 従来の演算バインディングよりも多くの 0-1 変数や制約式が必要となり, 適用できる問題の規模が制限される恐れもある. 種々の例に対する実験を通じて適用可能規模を評価するとともに計算量を削減する制約条件の与え方や, グラフ変形による解法などについて検討することが今後の課題である.

謝辞

本研究に際し, 常に御理解のある御指導, 御援助をいただきました白川功教授に心から感謝申し上げます.

また, ともに議論・検討をして頂きました本学の白川研究室の高橋瑞樹氏, 渡辺辰雄氏に深く感謝致します.

参考文献

[Aka93] H. Akaboshi and H. Yasuura: "COACH: A Computer Aided Design Tool for Computer Architects," *IEICE Trans. Fundamentals, Japan*, vol. E76-A, no. 10, pp. 1760–1769 (Oct. 1993).

[Lie95] C. Lien, P. Paulin, M. Cornero, and A. Jerraya: "Industrial Experience Using Rule-Driven Retargetable Code Generation for Multimedia Applications," *Proc. International Symposium on System-Level Synthesis (ISSS)*, pp. 60–65 (Sept. 1995).

[Mar95] P. Marwedel and G. Goossens: *Code Generation for Embedded Processors*, Kluwer Academic Publishers, 1995.

[Mar97] R. Leupers and P. Marwedel: "Retargetable Generation of Code Selectors from HDL Processor Models," *Proc. European Design & Test Conference (ED&TC) '97*, pp. 140–144 (March 1997).

[Pra96] J. Van Praet, D. Lannwwe, G. Goossens, W. Geurts, and H. De Man: "A Graph Based Processor Model for Retargetable Code Generation," *Proc. European Design & Test Conference (ED&TC) '96*, pp. 102–107 (March 1996).

[Sat94] J. Sato, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi and M. Imai: "PEAS-I: A

Hardware/Software Codesign System for ASIP Development," *IEICE Trans. Fundamentals, Japan*, vol. E77-A, no. 3, pp. 483–491 (March 1994).

[Min90] S. Minato, N. Ishiura and S. Yajima: "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 52–57 (June 1990).

[Yam97-1] M. Yamaguchi, A. Yamada, T. Nakaoka, and T. Kambe: "Architecture Evaluation Based on the Datapath Structure and Parallel Constraint," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC '97)*, pp. 503–508 (Jan. 1997).

[Yam97-2] 山口 雅之, 石浦 菜岐佐, 神戸 尚志: "組込みシステム向けリターゲッタブルコンパイラの方式," 信学技報 *VLD97-90, FTS97-53*, pp. 85-92 (Oct. 1998).

[Yam98] 山口 雅之, 石浦 菜岐佐, 神戸 尚志: "非直交なデータパスに対するリターゲッタブルコンパイラのバインディング手法," 第 *11* 回 回路とシステム *(*軽井沢*)* ワークショップ, pp. 481–486 (Apr. 1998).