論文 / 著書情報
Article / Book Information

論題(和文)	ディジタル信号処理回路の計算順序決定とそのシグナルプロセッサ用コンパイラへの応用
Title(English)	Computational Ordering of Linear Digital Networks and Its Application to a Compiler System for Digital Signal Processors
著者(和文)	杉野暢彦, 年清昭彦, 渡部英二, 西原明法
Authors(English)	NOBUHIKO SUGINO, Akihiko Toshikiyo, Eiji Watanabe, AKINORI NISHIHARA
出典(和文)	電子情報通信学会論文誌(A), Vol. J71-A, No. 2, pp. 327-335
Citation(English)	The Transactions of the IEICE, Vol. J71-A, No. 2, pp. 327-335
発行日 / Pub. date	1988, 2
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 1988 Institute of Electronics, Information and Communication Engineers.

論　文

ディジタル信号処理回路の計算順序決定とそのシグナルプロセッサ用コンパイラへの応用

准　員　杉野　暢彦[†]　　非会員　年清　昭彦[††]
正　員　渡部　英二[†]　　正　員　西原　明法[†††]

Computational Ordering of Linear Digital Networks and Its Application to a Compiler System for Digital Signal Processors

Nobuhiko SUGINO[†], *Associate Member,* Akihiko TOSHIKIYO[††], *Nonmember,*
Eiji WATANABE[†] *and* Akinori NISHIHARA[†††], *Members*

あらまし　ディジタルシグナルプロセッサ（DSP）を用いて信号処理回路を実現する場合，さまざまな処理に対して柔軟性があるが，プログラミングが非常に難しいという欠点がある．本論文は，ディジタル信号処理回路を容易に記述できる高級言語を提案し，これを用いた記述からDSP用にコードを生成するコンパイラについて述べている．この高級言語記述は非決定的に行い，コンパイラにおいて計算順序決定を行う．このような計算順序決定問題について最適なアルゴリズムは知られていないので，次のような発見的手法を提案している．まず，高級言語記述から木枝を用いた内部表現に変換する．これをいくつかの部分グラフに分割することにより，部分的に既存の最適コード生成アルゴリズムの適用を可能としている．部分グラフ間の計算順序は，深さ優先の探索およびそれを拡張した幅方向付き深さ優先の探索を用いる．本コンパイラを用いると，DSPのハードウェアの専門的な知識なしにプログラミングしても，生成したコードにある程度の効率が見込めることが，いくつかの例題により示されている．

1. まえがき

実時間ディジタル信号処理専用にディジタルシグナルプロセッサ（以下DSPと略す）LSIが発表され，広範囲に使用され始めている[(1)]．DSPはそのソフトウェアを変更することによって多種多様のディジタル信号処理に対処できるという利点をもっている．その反面，実行速度の高速性を重視したアーキテクチャを採用しているためソフトウェアの生成が非常に困難になっている．更に，プログラム全体の長さが信号処理システム全体の最大標本化周波数を決めるので，複雑な処理になるとプログラマへの負担が非常に大きくなってしまう．このような状況を改善するために高級言語からのコードの生成が必要と考えられる．

DSPプログラミングの高級言語化の試みとして今までに，C言語など比較的低級なプロトタイプ（汎用）言語からクロスアセンブルを用いた方法[(2),(3)]や，アセンブラに類似した独自の言語を用いた方法[(4)]等が開発されている．また，最近開発されたいわゆる第2世代DSPに対しては拡張したアセンブラなどが開発されている．以上のシステムは，効率の良いコードを生成可能であるが，プログラマがDSPのアーキテクチャと内部での信号の流れを考慮に入れる必要があり，ソフトウェア開発環境の本質的改善には至っていないと考えられる．そこで，プログラマがDSPのアーキテクチャにとらわれず，だれがプログラミングしてもある程度のコードの生成効率が見込めるようなコンパイラシステムが望ましい．また，DSP自体の歴史が浅いため，各社から発表されているDSPが多種類に渡り，こういった意味からもソフトウェアの生産性を悪化させている．DSPがまだ開発途上にあることを考えれば，

† 東京工業大学大学院総合理工学研究科，横浜市
The Graduate School at Nagatsuta, Tokyo Institute of Technology, Yokohama-shi, 227 Japan
†† NTTデータ通信事業部企画部，東京都
NTT Data Communication Secter, Tokyo, 100 Japan
††† 東京工業大学理工学国際交流センター，東京都
International Cooperation Center for Science and Technology, Tokyo Institute of Technology, Tokyo, 152 Japan

今後もその種類が増えていくと考えられ，コンパイラができるだけDSPの種類に依存しない方が良いと考えられる．

筆者らは，以上の二つのことを方針にしてDIMPL (Digital network IMPlementation Languageの略)[5]~[9]というシグナルフローグラフを記述する高級言語，および，この言語を用いたプログラムからターゲットDSP用にコード生成を行うコンパイラを開発してきた．このコンパイラでは，先に述べた二つの方針を満足させるために，ディジタル信号処理回路の計算順序を決定する処理が必要となる．一般にこの種の問題は計算機コード生成の立場からNP完全であることが証明されており，最適コード生成アルゴリズムは難しい[10]．そのため，ある限定条件下で最適コードを生成する方法，あるいは，近似最適解を求める方法(プログラマの発見的手法を含む)が考え出されているに過ぎない．そこで筆者らは，計算順序決定法として，最適コード生成法と近似最適解を求める方法の併用を提案し[5]~[7]，更にこれを拡張し効率の向上を図ってきた[8],[9]．本論文では，DIMPLコンパイラで用いた計算順序決定の方法について述べ，その効率について検討を加えている．

2. 回路の記述

まず，本研究で使用したコンパイラの入力記述について説明する．

ディジタル信号処理回路は，図1で示すような加算器 (Adder)，乗算器 (Multiplier)，遅延器 (Delay) の三つの基本要素を用いて，図2に示すようなシグナルフローグラフで表現することができる．図2は3次ラチスフィルタである．なお，図2中の小円は節点を表し，小円を囲む正方形は **3.3** で用いるマーキングである．本コンパイラでは，各要素をその入出力変数名を用いて，図1の下に示すように記述する．

全シグナルフローをコンパイラに入力する場合，図2のように回路の全節点に独立な名前付けを行い，記述する方法がある．これは，一般のシミュレーションや他のDSP用高級言語記述においてもよく用いられている方法である．一方，回路中で構造が同じ部分がある場合，その部分を「部分回路」として抽出して記述する方法も考えられる．図2の回路を例にとると，図3(a)のような，部分回路が抽出でき，全体の回路のシグナルフローグラフは図3(b)のようになる．これは，ちょうど汎用のプログラミング言語におけるサブルーチンの手法に相当する．DIMPLでは，以上述べたどちらの方法でも回路を記述することができる[5]．図3(b)の回路を後者の方法で記述した例を図4 (プログラム-1)に示す．主記述全体は宣言，部分回路の記述，部分回路を用いた全体の接続状況の記述の三つの部分から成る．FILTERによって主記述の開始を示す．続いてCONSTおよびNODE以下でそれぞれ乗算器の係数および使用する節点の宣言を行う．次にくる部分回路の記述はSUBCIRCUITで始まり，入出力

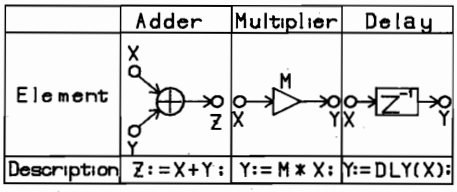


図1 ディジタル信号処理回路の基本要素とその記述方法
Fig. 1 The basic components of digital networks and their description.

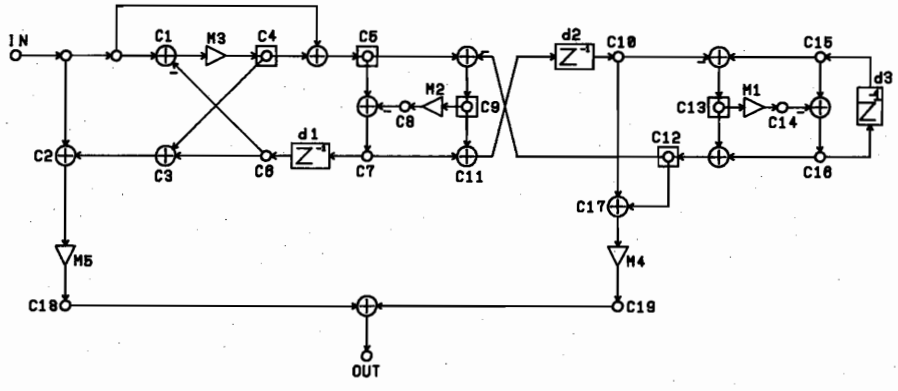
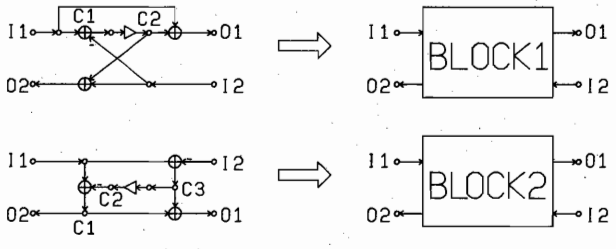
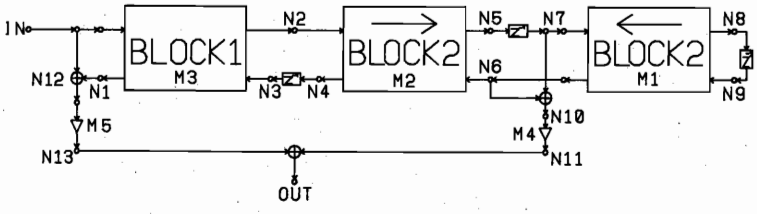


図2 シグナルフローグラフの例 (3次ラチス型)
Fig. 2 Example of signal flow graph (Third-order lattice network).



(a) Extraction of subcircuits



(b) Signal flow graph in Fig. 2 using the subcircuits

図3 部分回路を用いたシグナルフローグラフの表現
Fig. 3 Signal flow graph using the subcircuits.

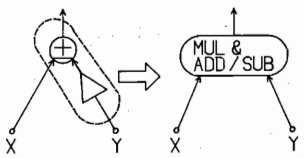


図5 積和演算の処理
Fig. 5 Multiplication and addition/subtraction operation.

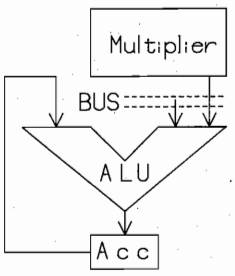


図6 DSPの一般的なアーキテクチャ
Fig. 6 General architecture of DSPs.

```
FILTER LATTICE2(IN > OUT);
CONST M1=0.123989,
      M2=0.165142,
      M3=-0.458306,
      M4=0.471563,
      M5=0.028438;
NODE  N1,N2,N3,N4,N5,N6,N7,N8,N9,
      N10,N11,N12,N13;

SUBCIRCUIT BLOCK1(I1,I2 > O1,O2 / M);
NODE C1,C2;
BEGIN
  C1:=I1-I2;
  C2:=M*C1;
  O1:=I1+C2;
  O2:=C2+I2;
END;

SUBCIRCUIT BLOCK2(I1,I2 > O1,O2 / M);
NODE C1,C2,C3;
BEGIN
  C1:=I1-C2;
  C2:=M*C3;
  C3:=I1-I2;
  O1:=C1+C3;
  O2:=C1;
END;

BEGIN
  BLOCK1(IN,N3 > N2,N1 / M3);
  N12:=N1+IN;
  N3:=DLY(N4);
  BLOCK2(N2,N6 > N5,N4 / M2);
  N7:=DLY(N5);
  BLOCK2(N9,N7 > N6,N8 / M1);
  N9:=DLY(N8);
  N10:=N7+N6;
  N11:=M4*N10;
  N13:=M5*N12;
  OUT:=N11+N13;
END.
```

図4 プログラム-1 (DIMPL記述例)
Fig. 4 Program-1 Example of DIMPL description in Fig. 3.

節点の他にパラメータとして乗算係数を引数に取ることができる．部分回路の記述は主記述と同じ構成であり，記述全体として入れ子構造になっている．ここではBLOCK 1およびBLOCK 2が部分回路である．

プログラミングが簡単に行えるという方針を反映させるために，DIMPLの記述では非決定的に記述を行えるようにしている．すなわち，プログラマは，通常の汎用プログラム言語のときのように前もって演算の順番について配慮する必要がない．このため，DIMPLコンパイラの処理において，与えられた基本要素間の結線情報を基にして計算順序の決定（後述）が必要となる．この決定の最適性がコード生成の効率をほぼ決める．

与えられたDIMPL記述は，コンパイラ内部では演算あるいは直接数値を表す節点，および，それらを結ぶ枝を用いて木の形で扱われる．これは，ちょうど一般のコンパイラでの算術式の表現方法に似ている．ここで，出力コードのダイナミックステップ数を小さく抑えるために，部分回路をその意味する回路構造で全体の回路に展開する[5]．すなわち，全体の回路は図3(b)から再び図2のような形に戻される．更に，DIMPLでは，乗算と加減算の組合せを図5に示すように積和演算という一つの非可換演算子で表すことにしている[6]．これは，一般にDSPが図6のように乗算と加算を連続して効率良く実行できるアーキテクチャを採用していて，これを効率良く利用するためである．

3. 計算順序の決定法

3.1 プレシデンスフォームからの計算順序決定

シグナルフローグラフから各節点の計算順序を求めるとき，プレシデンスフォーム(Precedence Form)[11]がよく用いられる．プレシデンスフォームとはシグナルフローグラフの節点をその値の決定の前後関係に従ってグループ分けしたものである．プログラマがDSPのアセンブラを作成する際やシミュレーション等で計算を行う際にも最低限その半順序関係を守る必要がある．例として図2の回路のプレシデンスフォームを図7に示す．しかし，プレシデンスフォームでは，節点をグループに分けるに過ぎず，同じグループに含まれる節点間や2グループを結ぶ枝の間には，計算の順序関係は与えられない．従って，同じグループに含まれる節点をすべて並列に実行できる場合以外は，最適な計算順序を選ぶために，プレシデンスフォームの示す半順序関係を崩さず，しかも効率の良い計算順序を決定するアルゴリズムが必要となる．

3.2 深さ優先の探索

ディジタル信号処理回路のシグナルフローグラフから遅延器をすべて取り除いたプレシデンスフォームは有向閉路を含まない有向グラフ（Directed Acyclic Graph：以下DAGと略す）となる（図7参照）．ここからすぐにプレシデンスフォームのグループごとに順番に計算する方法も考えられるが，メモリへのデータの退避等のオーバヘッドが頻繁に生じるため好ましくないと考えられる．そこで，グラフ的な結び付きを重視して，できるだけ局所的に計算順序決定を行った方が望ましい．このためにグラフのアルゴリズムの一つである「深さ優先の探索」[12],[5],[7]を利用して，有向グラフを局所的に探索し，節点の計算順序を決定する．

プレシデンスフォームに深さ優先の探索を用いれば，節点間に一応の計算順序関係を与えることができる．しかし，これはいかなる意味でも計算順序の最適性を保証していない．更に，深さ優先の探索では扱う節点の数が増えるにつれて探索時間が増す上に，探索時に複数節点の選択が頻繁に発生して，コード化した際に効率の悪い計算順序を決定する可能性が高くなる．

3.3 部分グラフへの分割

プレシデンスフォームで与えられた半順序関係を守りつつ，深さ優先探索を用いて計算順序を決定する場合には，3.2で述べたような欠点が生じてしまう．そこで，あらかじめ，結び付きの強い節点のグループを一つの部分グラフとして抽出して，その部分グラフごとに計算順序を決定した方が効率が良いと考えられる．

プレシデンスフォームから得られるDAGは，例えば図7のC6→C1→C4→C3→C6で示されるような無向閉路を含む一般的なクラスのDAGである．このようなクラスのDAGに関するコード生成問題はNP完全になることが証明されている[10]．一方，一般的に無向閉路を含まない2分木DAGに関してはインテジャーラベリング等を用いた最適コード生成アルゴリズムが知られている[12]．このアルゴリズムを利用するためには，もとのDAGから無向閉路を取り除き，2分木DAGにする必要がある．無向閉路は図7のC4, C5等に示すような，遅延器からの入出力を含まない分岐点が原因となって形成されている．そこで，この分岐点においてDAGを切り離すことによって無向閉路を取り除くことができる．図2においてはこの分岐点は正方形で囲んで示してある．切り離した結果，もとのDAGは図8に示すように多くの無向閉路を含まないクラスのDAG（部分グラフ）に分解される．なお図8

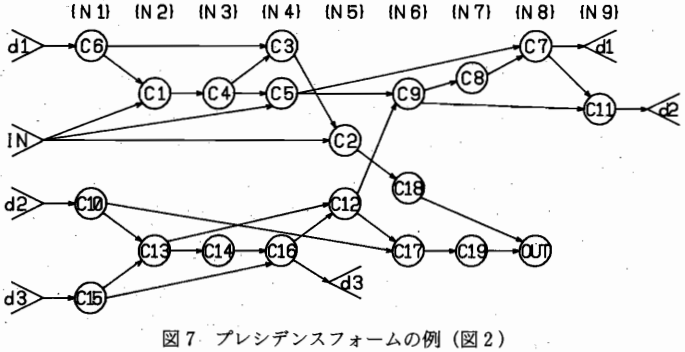


図7 プレシデンスフォームの例（図2）
Fig. 7 Precedence form for network in Fig. 2.

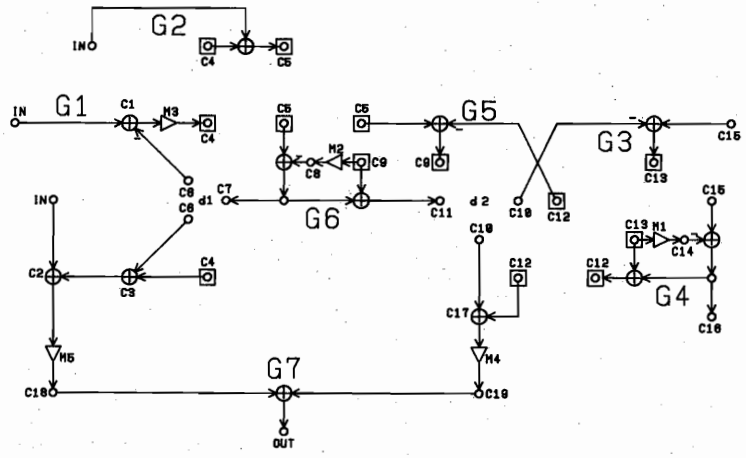


図8　部分グラフ分割例（図2）
Fig. 8　Partitioned graphs of network in Fig. 2.

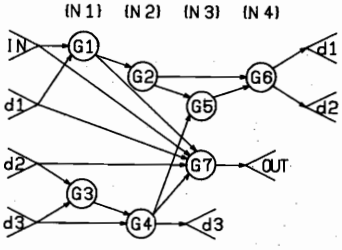
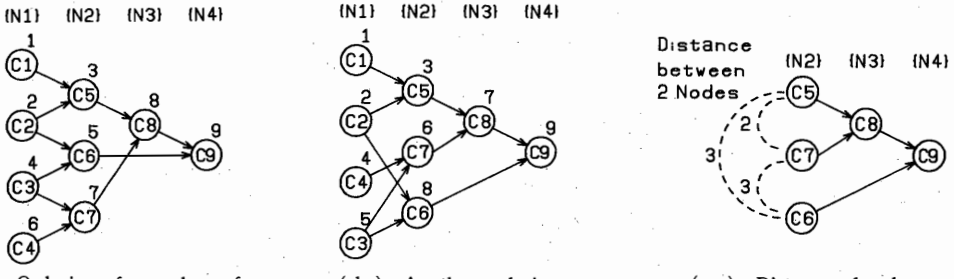


図9　部分グラフ単位のプレシデンスフォーム
Fig. 9　Precedence form for partitioned graphs in Fig. 7.



（a）Ordering of precedence form　（b）Another ordering　（c）Distance of nodes

図10　深さ優先の探索の問題点
Fig. 10　Problem in the depth first search.

中でG1,G2などは後で用いるために各部分グラフにつけられた名前を表す．また，もとの分岐点はちょうど図8のC4，C5等のように部分グラフにおけるDAGの葉（leaf）と根（root）になる．このようにして得られた部分グラフは簡単に2分木DAGに変形することができる．

以上のことをまとめると，部分グラフ分割の際，与えられたディジタル信号処理回路のシグナルフローグラフは次の2種類の節点において切り離されることになる．

① 遅延器の両端および入力節点
② 遅延器の入出力を含まない分岐点

これらの節点は，一標本化周期の間での参照がすべて終了するまでその値をメモリあるいはレジスタなどに一時保存する必要がある節点であり，ここで分割することはコードを生成する上においても都合がよい．

このようにシグナルフローグラフが部分グラフに分割された後，部分グラフ間についても計算順序を決める必要が生じる．得られた部分グラフ間には，改めて部分グラフ単位のプレシデンスフォームで示される計算の半順序関係が成立する．例えば，図8の場合については部分グラフ単位のプレシデンスフォームは図9のようになる．図7と比べプレシデンスフォームの規模が小さくなることがわかる．節点単位のプレシデンスフォームのときと同様，部分グラフ単位のプレシデンスフォームについても深さ優先の探索等を用いることによって計算順序を決める必要がある．しかし，前述したように深さ優先の探索についての最適性がいえないため，全体の効率は深さ優先の探索に依存している．

3.4　幅方向優先順位付き深さ優先の探索[8]

3.1で述べたように，深さ優先の探索中に，同時に複数の節点あるいは部分グラフが選択可能になったとき，どれから先に探索を始めるかによって，探索の最適性が左右される．例えば，図10(a)のようなプレシデンスフォームを考えてみた場合，同図に添えた数字

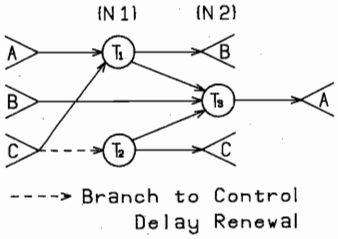


図11 遅延器更新枝
Fig. 11 Branch to control delay renewal.

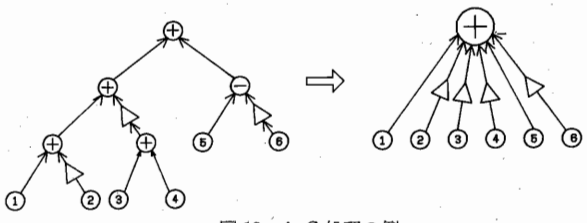


図12 A-C処理の例
Fig. 12 Example of A-C operation.

のように順序付けできる．ところが，図の{$N2$}グループを入れ替えた図10(b)のように順序付けもでき，この方が探索の効率が良いと考えられる．すなわち，図10(c)に示すようにプレシデンスフォームの{$N2$}グループ内で，節点間がグラフ的に近い関係にあるものから優先的に探索した方がよいと考えられる．そこで，節点間の優先順位の基準として節点関連度を用いこれをプレシデンスフォームのグループ内での優先順位として深さ優先の探索を行う[8]．これは，深さ方向だけでなく横（幅）方向も考慮していることになる．

3.5 遅延器のデータ更新

回路の計算順序を決定する際，遅延器データの更新が問題となることがある．一標本化周期内での遅延器の参照がすべて終了するまでに遅延器の値が更新されてはならない．例えば，いま遅延器の入出力の関係を含めたプレシデンスフォームが図11のように与えられたとする．なお，図11中のT_1～T_3は節点あるいは部分グラフのどちらかを表す．遅延器A，Bのように遅延器入出力の間に明確な順序関係が存在している場合は問題はない（Bの場合には値の一時退避が常に必要）．遅延器Cのように節点の計算順序の選び方によって遅延器の入出力の順序関係が左右される場合には，探索時の順序決定で指針となるように図11の点線で示す遅延器制御枝[8]を設けて遅延器の入出力の間に明確な順序関係を挿入し，遅延器更新値の余分な一時退避を避けることができる．

以上は各遅延器に1面のメモリを割り当てる場合を想定しているが，2面以上のメモリを標本化時間ごとに切り替えて用いれば，上述したような問題は生じない．

3.6 部分グラフの多分木への展開

いままでのところの問題は，入力したディジタル回路の構造を変えずに，いかにして最適な計算順序を決定するかにあった．ところで，DSPは一般的に図6に掲げたようなアーキテクチャをもち，積和演算に都合が良い．そこで，DSPにとって都合の良い方法で計算を行わせることを考えてみる．**3.3**で述べた部分グラフ分割後，各部分グラフは2分木DAGの形をしている．これらに図12に示したように多分木化の処理を施す．この処理は文献(13)の結合法則(Associative law)と交換法則（Commutative law）に基づく処理を利用しているのでA-C処理と呼ぶことにする(処理について詳しくは文献(9)を参照)．この処理の結果各部分グラフは図12の右の図に示すような多分木となり，これは式化すれば，

$$Y=a_1x_1+a_2x_2+\cdots+a_nx_n \qquad (1)$$

のように積和形式で表すことができ，DSPでの演算に有利であると考えられる．状態空間構成の回路は，同じように積和形式で表されていて，DSPでの実現に適しているが，全体の演算量が多いという欠点がある．ここで用いた方法は部分グラフ内部に適用するもので，多分木化の後も共通部分式をまとめるという分岐点の効果は残しており，演算全体の量は状態空間構成より軽減される．

しかし，この処理を行うことによって，もとのディジタル回路の構造が変化してしまうので，オーバフロー，リミットサイクル，感度等といったフィルタの諸特性に好ましくない影響を及ぼす可能性も考えられる．特に，固定小数点方式のDSPへの影響は顕著と予想される．

3.7 部分グラフ再結合

3.3で述べた部分グラフ分割も分割が細かくなり過ぎるとディジタル回路の全節点数が増加したときと同様，深さ優先の探索の最適性がなくなる等，問題になることがある．そこで部分グラフとしては最適コード生成アルゴリズムが適用できる範囲で大きいクラスのDAGの方が望ましい．2分木DAGだけでなく，DAG中に無向閉路を高々一つ含むクラスのDAG（$\mathcal{L}$-DAG）についても，既に2分木DAGと似た方法の最適コード生成アルゴリズムが発表されている[14]．しか

し，このアルゴリズムを適用するためには，部分グラフの中に$\mathcal{L}$-DAGが存在しなければならない．2分木DAGのときのようにもとのDAGから部分グラフ分割によって直接このクラスのDAGを得るのは困難である．そこで，既に分割されている部分グラフを$\mathcal{L}$-DAGの性質を満たすように再結合することが考えられ，これにより，ある程度の効率向上が得られている[8]．

4. コンパイラの構成とコンパイル例

以上で述べた方法を用いてコンパイラシステムを構築すると図13に示したような構成となる．コンパイラはまずDIMPL記述の入力ファイルを読み込み，構文解析（Syntax Analysis）を行い，コンパイラ内部で用いる中間コード（Intermediate Code）として枝のデータを生成する．次に本論文で取り扱った方法を用いた計算順序決定（Decision of Order）を行う．ここまでの部分はターゲットのDSPに依存しない部分である．最後に，決定した順序関係に従ってコード生成（Code Generation）を行いアセンブラのソースコードを出力する．

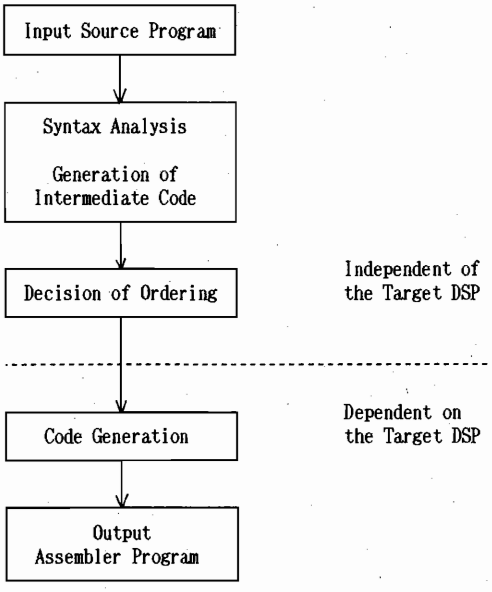


図13　コンパイラの構成
Fig. 13　Structure of the compiler.

実際にコンパイラを当初M-280大型コンピュータ上でPASCALを用いて記述した．現在では，PC 9801パーソナルコンピュータにも移植している．実行速度はPC 9801上でも数十秒（プログラム-1の場合）程度であり，十分実用に耐えると考える．

ターゲットDSPとしては日本電気製のμPD 7720[15]を使用した．前述した各方法を比較するために，各方法を加えたコンパイラで同じ回路記述を別々にコード生成させてみた．その結果を人間が生成したコード長と共に表1に示す．表中，WAVE型3次はDSPについて十分な知識のない人がハンドアセンブルしたものである．回路が複雑でもありコードはコンパイラに比べて長くなっている．この他の例に関してはハンドアセンブルに比べて1.1～1.6倍程度に収まっている．また，ディジタルフィルタの種類によっては探索或は計算順序決定の手法の変更によりコード生成効率が改善されていることがわかる．最終的にはコード生成効率は表1に挙げなかったものも含めて1.5倍以内に収まっている．

例として，実際にコンパイラを用いてプログラム-1について生成したコードを図14（プログラム-2）に示す．プログラム中で各インストラクション後の/*と*/で囲まれた部分（コメント）は，そこにおいてメモリから参照される節点の名前等を表す．

以上より，ディジタルフィルタ回路に対して本コンパイラは，ある程度効率の良いコード生成することがわかる．また，人間がこのコンパイラの生成した結果

表1　計算順序決定法の改善によるコンパイラ生成コード数の推移

使用したＬＰＦ	ＤＩＭＰＬコンパイラ			ハンドアセンブル
	部分グラフ分割＋深さ優先の探索	部分グラフ分割＋幅方向付き深さ優先の探索	A-C処理付加	
WAVE型3次	47	44	43	55
2D縦続6次	58	51	48	48
WDLF 7次	61	55	55	46
Lattice 5次	49	42	41	30
State Space 3次	43	33	33	—

```
/*              ASM7720 source list                 */
/*                                                  */
/*              filter name :LATTICE3               */
/*                                                  */
/*              date  -----  87/ 6/26               */
/*              time  -----   2: 5:17               */
/*                                                  */
MPROG;
     ORG OOOH;
     LDI OOOH,@SR;
     LDI OOOH,@DP;
SETI:LDI OOAH,@RP;
     /* ---- A/D converter interface ---- */
IN   :JNSIAK IN;
     OP MOV SIM,@A;
     /* ---------------------------------- */
     /*                                    */
     OP                    M1       A   ,@MEM /* LATTICE2  IN        0   1 */;
     OP SUB ACCA,IDB          RPDEC A   ,@KLR /*                        2 */;
     OP ADD ACCA,M         M3 RPDEC MEM,@KLR /*                        3 */;
     OP ADD ACCA,M                           /*                        4 */;
     OP                    M2       A   ,@MEM /* BLOCK1    C2        1   5 */;
     OP ADD ACCA,RAM       M3                /* LATTICE2  IN        0   6 */;
     OP                    M7       A   ,@MEM /* LATTICE2  N2        0   7 */;
     OP                    M1       MEM,@B    /* LATTICE2  N9        0   8 */;
     OP SUB ACCB,RAM       M3                /* LATTICE2  N7        0   9 */;
     OP                    M2       B   ,@MEM /* BLOCK2    C3        3  10 */;
     OP                    M2       MEM,@B    /* LATTICE2  N9        0  11 */;
     OP                    M2 RPDEC MEM,@KLR /* BLOCK2    C3        3  12 */;
     OP ADD ACCB,M                           /*                       13 */;
     OP                    M2       B   ,@MEM /* LATTICE2  N9        0  14 */;
     OP ADD ACCB,RAM       M1                /* BLOCK2    C3        3  15 */;
     OP SUB ACCA,IDB                B        /*                       16 */;
     OP                    M4       A   ,@MEM /* BLOCK2    C3        2  17 */;
     OP                    M4       MEM,@A    /* LATTICE2  N2        0  18 */;
     OP               DPINC M7 RPDEC MEM,@KLR /* BLOCK2    C3        2  19 */;
     OP ADD ACCA,M                           /*                       20 */;
     OP               DPDEC M7       A   ,@MEM /* BLOCK2    C1        2  21 */;
     OP ADD ACCA,RAM       M5                /* BLOCK2    C3        2  22 */;
     OP SUB ACCB,IDB          RPDEC B   ,@KLR /*                        23 */;
     OP ADD ACCB,M         M3 RPDEC MEM,@KLR /*                        24 */;
     OP ADD ACCB,M         M1 RPDEC MEM,@KLR /*                        25 */;
     OP ADD ACCB,M         M5 RPDEC MEM,@KLR /*                        26 */;
     OP ADD ACCB,M    DPINC M4 RPDEC MEM,@KLR /*                        27 */;
     OP ADD ACCB,M                           /*                        28 */;
     OP                    M1       B   ,@MEM /* LATTICE2  OUT       0  29 */;
     OP               DPDEC M1       MEM,@B    /* BLOCK2    C1        2  30 */;
     OP                    M4       B   ,@MEM /* LATTICE2  N3        0  31 */;
     OP               DPINC M4       A   ,@MEM /* LATTICE2  N7        0  32 */;
     OP               DPDEC M1       MEM,@A    /* LATTICE2  OUT       0  33 */;
     /*  total  33 steps  except I/O and LDI operation */
     /* ---- D/A converter interface ---- */
     LDI 8OOOH,@TR;
     OP XOR ACCA,IDB                TR;
OUT :JSOAK OUT;
     OP MOV A,@SOM;
     JMP SETI;
EOF;
```

図 14　プログラム-2　プログラム-1 からのコード生成例
Fig. 14　Program-2　Example of μPD 7720 code generated from Program-1.

を参考にして，更に効率のよいコードを作成することもできる．

5. むすび

本研究ではディジタル信号処理回路記述用言語 DIMPL を開発し，その記述から DSP 用にコードを生成するコンパイラを試作した．コンパイラ内部で用いるディジタル回路の計算順序決定のための手法を提案し，それらの効率について検討を行った．その結果，直接節点のプレシデンスフォームの探索を行うよりも，ある小部分ごとに最適コード生成が行えるように部分グラフ分割を行い，その後深さ優先の探索を行った方が効率が改善できるという結論を得た．また，深さ優先探索を幅方向にも拡張し，探索効率を高めた．更に，DSP に都合が良いように計算方法を最適化することも試みた．これは，もとのディジタル回路の構造にある程度関係するが，コード生成効率の改善につながることを確認した．

本コンパイラを用いると，DSP に関する専門的知識がなくとも，ある程度効率の良いコードを得ることができる．また，本コンパイラはパーソナルコンピュータ上でも動作し，DSP 用のプログラム開発を非常に容

易にする．

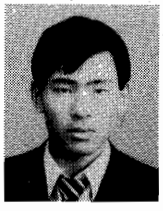
本コンパイラの問題点としては，もとのディジタル回路の構造によっては，部分グラフ分割時に多くの細かなグラフに分割されてしまう場合があり，他の探索或は計算順序決定の手法が十分に働かず，結果として全体のコード生成効率が悪化することが挙げられる．これについては部分グラフ分割の方法に問題があるのではないかと考えられ現在検討中である．また，現在発表され始めた第2世代のDSP用にも，ここで用いた計算順序決定の方針を用いてコンパイラを作成することを現在検討中である．

謝辞　本研究は坂本治之氏の東京工業大学大学院在学中の研究に端を発している．彼の初期の成果とプロトタイプのコンパイラの貢献するところは大きく，ここに深く感謝する．また，有益な御助言を頂いた東京工業大学の柳沢健教授，藤井信生助教授に感謝する．更に，貴重な資料の御提供や御討論を頂いた日本電気マイクロコンピュータ技術本部システム部の鈴木宗一氏，川上雄一氏，田中秀夫氏，および同社半導体応用技術本部マイクロコンピュータメモリ技術部藤高一郎氏に深く感謝する．

文　献

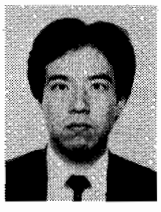
(1) 辻井重男："シグナルプロセッサとその応用"，コンピュートロール，15，コロナ社（昭61-07）．
(2) 小野，金山："信号処理プロセッサプログラム開発サポートシステムの構成"，信学技報，**CS83**-185（1984-03）．
(3) 小野，金山："信号処理プロセッサプログラム開発サポートシステムIO（イオ）の概要"，信学技報，**CS84**-34（1984-06）．
(4) O. Hyvarinen, M. Kylanpaa, J. Oksa and J. Skytta: "A Network Description Language (NDL) for digital signal processing", Proc. ECCTD 85, pp. 420-423 (Sept. 1985).
(5) 坂本，渡部，西原："シグナルプロセッサのためのディジタルフィルタ記述言語"，信学技報，**CAS84**-48（1984-06）．
(6) 坂本，渡部，西原："シグナルプロセッサ用ディジタルフィルタプログラムの最適化"，信学技報，**CAS84**-182（1985-01）．
(7) H. Sakamoto, E. Watanabe and A. Nishihara: "Code optimization of digital networks for signal processors", Proc. Int. Symp. on Circuits and System, pp. 1403-1406 (June 1985).
(8) 年清，渡部，西原："ディジタルシグナルプロセッサ用計算順序最適化"，信学技報，**CAS86**-31（1986-05）．
(9) 杉野，西原，渡部："第二世代DSP用コンパイラシステム"，信学技報，**CAS87**-4（1987-04）．
(10) J. Bruno and R. Sethi: "Code generation for a one-register machine", J. ACM, **23**, 3, pp. 502-510 (July 1976).
(11) R. E. Crochiere and A. V. Oppenheim: "Analysis of Linear Digital Networks", Proc. IEEE, **63**, 4, pp. 581-595 (April 1975).
(12) A. V. Aho, J. E. Hopcroft and J. D. Ullman: "アルゴリズムの設計と解析（I）"，pp. 158-170，サイエンス社（昭56）．
(13) R. Sethi and J. D. Ullman: "The Generation of Optimal Code for Arithmetic Expressions", J. ACM, **17**, 2, pp. 715-728 (Oct. 1970).
(14) 木村春彦："直列形プログラムのコード最適化一多レジスタ機械におけるアルゴリズムの複雑さ一"，信学論(D)，**J66-D**，2，pp. 167-174（昭58-02）．
(15) NEC："μPD 7720 シグナルプロセッサユーザーズマニュアル"（昭55）．

（昭和62年6月30日受付，8月21日再受付）



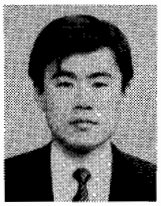
杉野　暢彦

昭62東工大・工・電気電子卒．現在同大大学院総合理工学研究科物理情報工学専攻修士課程に在学中．ディジタル信号処理用プロセッサのハードウェア，ソフトウェアに関する研究に従事．



年清　昭彦

昭59東工大・工・情報卒．昭61同大大学院電子物理・修士課程了．同年，NTT入社．修士課程在学中，ディジタルシグナルプロセッサの最適コード自動生成の研究に従事．



渡部　英二

昭56電通大・電気通信・電波通信卒．昭58同大大学院修士課程了．昭61東工大大学院理工・電子物理博士後期課程了．工博．同年同大学院総合理工・物理情報助手．ディジタルフィルタを中心に離散時間回路網の構成と実現の研究に従事．

西原　明法

昭48東工大・工・電子物理卒．昭53同大大学院博士課程了．工博．同年より同大勤務．現在，同大理工学国際交流センター助教授．東南アジア諸国との学術交流事業のほか，回路理論，電子回路等の研究，教育に従事．信号処理と信号処理用プロセッサにも興味をもつ．IEEE会員．