ソフトウェア工学 90－1
（１９９３．２．４）

計算グラフによるＤＳＰのための ディジタルフィルタプログラム自動生成

北岡 信一　加藤 恭子　石端 尚正　武部 幹
金沢工業大学　リコー電子デバイス事業部　金沢大学 工学部

ディジタルフィルタ専用のディジタルシグナルプロセッサ(DSP)用コンパイラを開発した。ディジタルフィルタの動作を記述する連立差分方程式を入力し、計算木／グラフを作成する。次に実行時間の短縮化の観点から、サンプル値変数を遅延させる処理を最適化する。またデータバス使用率、パイプライン密度を向上させるため、木／グラフを変形し、コード生成を行い、最後に差分方程式木の順序決定法を述べている。

Auto-Programming onto a DSP for Digital filters using Computational Graph

Shinnichi KITAOKA　Kyoko KATO　Naomasa ISHIBATA　Tsuyoshi TAKEBE
INFOR. and COM. Engineering　RICOH　Faculty of Engineering
KANAZAWA Institute of Technology　Electronics Development Center　KANAZAWA University

Special DSP compiler for digital filter is developed. A difference equation system which describes digital filters behavior is inputed and computation tree/graphs are determined. From the view of shortenning the execution time, operations for delaying sampled values is optimized. To enhance data sub use-rate and pipline density, computation tree/graphs are modified, from which code is generated. Finally, computation ordering in difference equation trees is desribed.

１．まえがき

ディジタルシグナルプロセッサ（ＤＳＰ）は、実時間信号処理を可能とするため、演算高速化のための独特のハードウエアアーキテクチャを持つ。すなわち、プログラムメモリとデータメモリとの分離、複数の内部バス、パイプライン、積和演算命令の高速実行演算機構等を持つ。汎用プロセッサでは多様なアドレッシング方式を持ち、メモリアクセスに関する機能は充実しているが、ＤＳＰでは高速化のためこの機能は十分ではない。ＤＳＰの機能を活かした効率よい実行コードを得るにはプログラミングに経験と技術を要する。ディジタル信号処理の多くの分野への浸透に伴いＤＳＰの利用は広がっている。手軽にＤＳＰを使うための開発環境の向上のため、コンパイラ言語が提供されるようになった[1][2]。

ＤＳＰを対象としたコンパイラは、汎用高級言語をＤＳＰに実装するものと、ディジタル信号処理を前提にしたものの２つに分けられる。前者はＣ言語に代表される高級言語を提供し、あらゆるプログラム開発を容易にするが、その代償として、実行プログラムの速度、メモリ消費量などの点の品質は優れない。一方後者は専用コンパイラと言え、開発するプログラムの用途に応じたコンパイラを容易するので高品質な実行プログラムを得ることができる。

本研究の目的は、ディジタルフィルタを対象とする専用コンパイラの開発にある。ディジタルフィルタプログラムの生成法に関する研究には、フィルタ回路のシグナルフローグラフを記述する言語を用い、これをプレシデンスフォーム(PRF)に変換し、実行コードを得るものがある[3]。

本文は、フィルタ回路の加算器での差分方程式からリコーＤＳＰ RP5C71/72上で実行可能なディジタルプログラムコード（以下実行プログラムと称す）を自動生成する方法を述べる。本方法は、実行プログラムをソフトウェア、ハードウェアの両面から最適化する。コード生成は、計算グラフを用いＤＳＰ特有の命令体系を活かす。ハードウエアに対する最適化を、データバスとパイプラインの使用効率の向上について行う。さらに、連立差分方程式の計算順序決定に、計算グラフ[4]を用いる。

以下、ＤＳＰとそのプログミングの背景、実行プログラム生成法、連立差分方程式の計算順序決定法、実験の結果の順に述べる。

２．ディジタルシグナルプロセッサ

ＤＳＰは高速演算を求められるが故に、個性のあるアーキテクチャが多種多様に存在する。従ってＤＳＰを活用するとき、そのアーキテクチャを熟知した上でハードウエアを活かすアセンブリ言語プログラミング技術が必須となる。以下に、ＤＳＰのプログラミングでの注意点と,5C72の特徴について述べる。

２．１ ＤＳＰのプログラミング

ディジタルフィルタのプログラムをコーディングするとき、回路ブロック図、シグナルフローグラフ(SGF)、差分方程式のいずれかが与えられる。

ブロック図、ＳＦＧに基づいてプログラムをコーディングする場合、乗算、加算、遅延の各演算のつながりによって演算の計算順や変数の記憶域配置を決定する。この時プレシデンスフォームなどを利用することもある。これに対し差分方程式による場合は、遅延の順に注意しながら計算式として差分方程式をコーディングする。

アセンブリ言語により、パイプライン、メモリバンク、マイクロコーディングなどのＤＳＰ固有のハードウエアに合わせたプログラムを作る際の注意について述べる。

メモリ上へのデータ配置は計算順序、命令コードの決定同様コーディングにおける重要な要素で、一般にデータ配置について目的のＤＳＰに合わせるよう考慮したプログラムは、しないものに比べて実行速度が向上する。

ディジタルフィルタにはディジタルシステム特有の遅延器 z^{-1}が含まれ、これを実現するには、あるサンプル時点の変数x(n)を次のサンプル時点でx(n-1)として扱う処理が必要である。これを遅延処理と呼ぶ。遅延処理の方法には、データを指し示すポインタを移動する方法と、データ領域の

データをロードとストアで移動する方法がある。ポインタ移動法では連続したメモリ領域に変数を整列配置しなければならない。一方ロードストア法を用いる場合は、ＤＳＰでは演算とメモリアクセスは共に処理時間は等しいことに注意して、実行ステップ数が増加しないようにプログラムをコーディングする必要がある。

２．２ リコーRP5C71/72

リコーＤＳＰ RP5C71/72（以下5C72）は、次の特徴を持つ。複数内部バス構成、２つの独立したインデックスユニット、データメモリ（内臓デュアルポート RAM）とプログラムメモリとの分離、４段パイプラインアーキテクチャの採用により、命令フェッチと２つのオペランドフェッチの３メモリアクセスを１クロックサイクルで完了する。さらに、メモリアクセスと演算とのパイプライン動作により、演算命令の実行と次の演算のデータロードまで含め１クロックサイクルで完了する EDUALアドレッシングモードを持ち、最大10MOPS(Million Operation Par Sec.)の演算能力を有する。

EDUALモードでは同時にアクセスする２つのデータをそれぞれのインデックスユニットで指す必要がある。

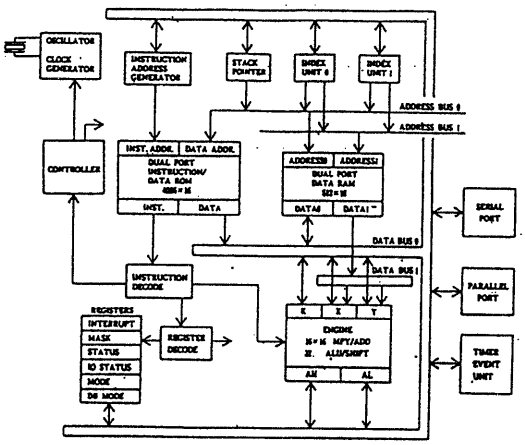


図 1: RP5C71/72 ブロック図

また、ロード、ストア命令は演算命令に等しい処理時間を必要とし、バスの使用率を下げると同時に、パイプラインの各ユニットをアイドル状態にするのでなるべく使わないことが望ましい。

３．ＤＳＰプログラムの自動生成

ディジタルフィルタを表現する差分方程式からのアセンブラプログラム自動生成法を述べる。

本生成法の特徴は次の２点にある。

(1) 連立差分方程式の計算グラフとＤＳＰ命令の計算木を用いてコード生成を行い、最適化処理についても木を用いる。

(2) アセンブラでプログラムをコーディングする場合のノウハウにより5C72に依存する最適化を行う。

ここで計算木とは計算式を２分木で表現したもので3.1で述べる。計算グラフ[4]とは計算木を連結したものをいい、3.5に述べる。

プログラム自動生成は次の４部、

・計算木／計算グラフの作成
・遅延処理最適化
・ＤＳＰ依存最適化
・コード生成

からなり、以上を3.1から3.4で述べ、3.5で連立差分方程式の計算グラフによる式の木の計算順序決定法を述べる。

３．１ 計算木の作成

差分方程式を計算木で表現する。ここで、計算木は、各節を演算子とし、葉は被演算子とする。また、根は常に等号になるが便宜上図２のように代入される変数を最上位に置き、これを代入先と称し等号の節を省略し表記する。

引き算の部分木では、引く変数、引かれる変数の方向を決め区別する。例として式(1)の差分方程式は、LISP表記すると、式(2)となり、図２の計算木を得る。

次に、ＤＳＰの演算命令を計算木で表す。演算命令を、変数にレジスタを使った式と考え同様に計算木にする。図３は、5C72におけるMAC（積和）命令の計算木の例である。

以下、差分方程式の計算木を差分方程式の木、演算命令の計算木を命令の木とする。

(1)

$$(= y[n](-(+(*a_1\, x[n])(*a_2\, x[n-1]))(*b_1\, y[n-1]))) \quad (2)$$

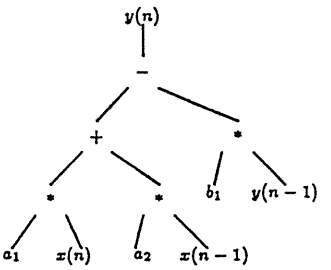


図 2: 式 (1) の計算木

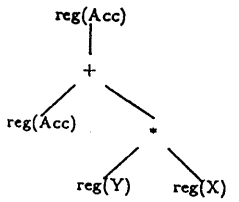


図 3: MAC 命令の計算木

3.2 遅延処理の最適化

遅延処理の方法には2.1でのべたように、ポインタ移動法と、ロードストア法がある。ポインタ移動法は遅延の量が多いとき威力を発揮するが、多くの変数を扱いづらくデータを記憶域に整列して置かなければならないなどの欠点がある。5C72ではポインタ設定、移動などインデックス機能に制限が多いのでロードストア法を採用する。

しかし、ロードストア法は1つの変数の遅延処理に原則的にロード命令、ストア命令を各1回実行する必要がある。5C72ではロードストアはなるべく避けたい命令なので、遅延処理に係わるロード命令を削減するために、遅延する変数に関する演算の計算時期を制御し、演算用ロード命令に遅延処理用ロードの役目を兼ねさせる。即ち、遅延処理は式の計算後、一括して実行するのではなく、式の計算過程の中に散りばめる。遅延処理を計算過程の中に組み込むと、遅延する変数の生存期間が重要になる。　例えば、一つの式の計算過程でx(n-1)をロードする演算があり、後で遅延処理のためにこのx(n-1)をx(n-2)の記憶域にストアしたい。そのためには、このストア命令は、このサンプル点での処理におけるx(n-2)を参照する全計算を終えた後に実行する必要がある。一般に変数の遅延量に従ってロード、ストア命令実行時期を制御する。

実行時期の制御は差分方程式木の加（減）算順序の入れ替えにより行う。並べ替えの順は最も古い（遅延量の多い）変数x(n-D)を含む加算の部分木を、木の最下位レベルに置き、x(n-0)まで順に上位に向かって並べる。例えば、図4に示すように、遅延量の少ない変数から遅延量の多い変数に向け矢印を付加し、この矢印が全て一方向になるよう部分木を入れ換える。図5は演算命令シーケンス中に遅延処理のためのストア命令が挿入され、ロード命令が削除されている様子を示す。

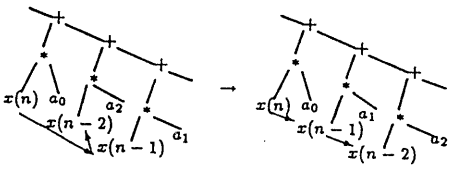


図 4: 部分木の交換

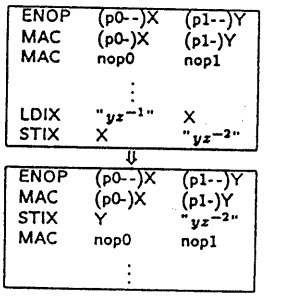


図 5: 遅延処理における LDIX 命令の削除

3.3 ＤＳＰに依存する最適化

ＤＳＰのように特殊なハードウエアアーキテクチャを持つプロセッサでは、各機種固有の最適化が必要不可欠である。本研究では、5C72を使うアセンブラプログラミングでの経験的な最適化を採り入れる。5C72では

(1)演算に必要なデータを、2メモリバンクのそ

れぞれに、同時にロードしたいデータを振り分け配置し、1サイクルで2データを逐次連続してロードする。

(2) 4段パイプラインの各ユニットの連続動作を確保する。

この2点が5C72の性能を発揮させるポイントである。これらはそれぞれ、データバス使用率の向上、パイプライン密度の向上についての最適化といえる。

3.3.1 データバス使用率の向上

EDUALアドレッシングモードを使うために、(1)のようにデータを各メモリバンクに配置し、同時に読み出すデータを2個のインデックスユニットのポインタで別々に指す。その結果、データバスの使用率が向上する。

データ配置を次のように行う。2項演算で同時に必要なデータを干渉グラフ[5]で明らかにし、別々のメモリバンクに配置する。即ち、積の項では乗数、被乗数グループに、また、和、差の項を同様にグループ分けする。

これは木を用いて次のように表現できる。葉の親から見て、左と右のグループに全ての葉を分ける。それぞれのグループをまとめ図6のように各メモリバンクに配置する。

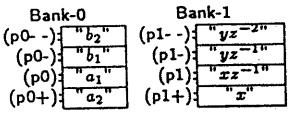


図 6: メモリバンクデータ配置

ENOP	(p0--)X	(p1--)Y
MAC	nop0	nop1

図 7: EDUAL モードアドレッシング

各メモリバンクに配置したデータのロードは、図7に示すようにENOP命令を使って行う。ENOP命令は、演算器が停止し、インデックスユニットなどが動作する命令で、EDUALモードアドレッシングのみを使用できる。

3.3.2 パイプライン密度の向上

3.3.1で、1サイクル当たりのデータバスの使用率は向上したが、ここでは、プログラムシーケンス全体でのデータバス使用率の向上をはかる。それはパイプライン密度を上げられることによる。パイプライン密度とは、パイプラインの各ユニットのアクティブ状態、アイドル状態の比率を表す。あるユニットがアイドル状態になると、パイプライン密度は低下する。パイプライン密度の向上は、命令シーケンスにおけるENOP命令とそれ以前の演算命令との統合により実現する。即ち、ENOP命令で指定するロードデータパラメータを、1ステップ以上前の演算命令で使われていないバスサイクルに組み込み圧縮する。図8ではMAC命令の未使用バスサイクルnopに組み込んでいる。

ENOP	(p0--)X	(p1--)Y
MAC	nop0	nop1
ENOP	(p0-)X	(p1-)Y
MAC	nop0	nop1
ENOP	(p0)X	(p1)Y
MAC	nop0	nop1

⇓

ENOP	(p0--)X	(p1--)Y
MAC	(p0-)X	(p1-)Y
MAC	(p0)X	(p1)Y
MAC	nop0	nop1

図 8: ENOP の圧縮

3.4 コード生成

式のコード生成は、逆ポーランド記法でスタックを利用するもの、一般的なコンパイラの内部表現の3つ組、4つ組によりデータロードと演算命令を生成するものが代表的である。本システムでは、計算木によりＤＳＰ特有の積和演算などを有効に利用するコード生成を行う。

本コード生成法は、差分方程式木の部分木に適合する命令木を探索する。探索の結果、差分方程式木の部分木を計算するための演算命令が選ばれる。

次に、命令木の結果の代入先のレジスタ名を、差分方程式木の選んだ部分木を取り除いた位置に接続する。これによりレジスタにデータを生存させ、次の演算に使用し、メモリアクセスを避けることができる。

選ばれた演算命令の演算データのロード命令を、取り除いた部分木と命令木の照合によって生成する。即ち、命令木の被演算レジスタに、部分木の変数をロードする命令を生成する（図９参照）。このとき、３．３．１で述べたように、あらかじめEDUALモードっを使うようになっているので、ロード命令はENOP命令のEDUALモードにより行う。

この操作を差分方程式の木がレジスタ名だけになるまで繰り返し行う。

ここで、差分方程式木から部分木を選ぶ方法が問題となる。本稿では、木の最下位レベルから順に選んでゆく方法をとる。さらに、選ぶ部分木の大きさも問題である。ＤＳＰでは命令木の大きさが一定ではない。命令木の大きい命令を選んで行けば、当該差分方程式木にたいするコードのステップ数は少なくて済む。よって、大きさ順の優先順位を命令木につけ探索に用いる。これらを合わせてコード生成法をまとめると、

(1)命令木を優先順位に従い選ぶ。
(2)命令木に等しいシルエットの差分方程式木の部分木を検索する。
(3)部分木をレジスタ名に付け変える。
(4)データロード命令(ENOP命令）を生成。
(5)(1)に戻り差分方程式木がレジスタ名になるまで繰り返す。

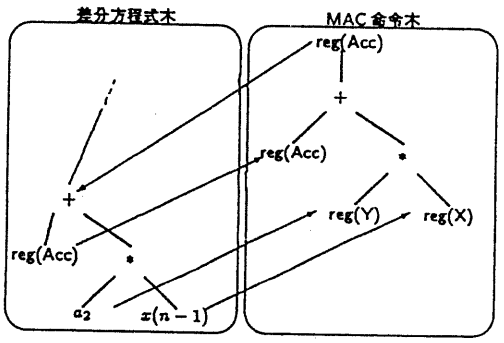


図 9: ロード命令生成

ロード命令生成後パイプラインの最適化を行うため、３．３．２で述べたように命令シーケンスにおけるデータロード命令の圧縮を行う。

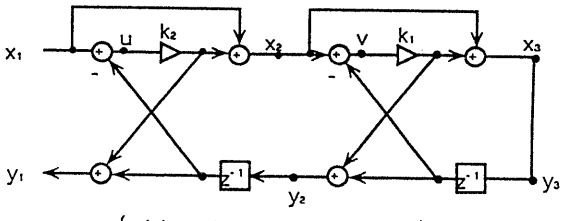
３．５ 連立差分方程式のコード生成

これまでは、単一の差分方程式の場合について考えてきた。ここでは、差分方程式が複数の場合、即ち、連立差分方程式のコード生成を行う。

コンパイラは一般に、いくつかの式が記述されている場合、記述された順に式のコード生成をする文法を採る。しかし本研究では、連立する差分方程式木が互いに独立ではなく依存関係を持つので、記述順にコード生成すると正しい出力が得られないことが生じる。このようなことの無いように、自動的に差分方程式間の計算順を求める必要がある。以下に、計算グラフを用いた差分方程式木の計算順序決定法を述べる。

ある差分方程式木の計算過程に別の差分方程式木の評価を必要とす場合、前者の木の葉の中に、後者の根に相当する変数が存在する。そこで、差分方程式木の間で、互いにその根と葉の変数が等しものを連結する（図１１（ａ）参照）。連結の結果、計算グラフとなる。

さらに、遅延の大きい木の処理を優先することを考慮するため変数ごとに遅延の少ないものから、多いものへと矢印を付ける。その矢印が連結以前の二つの差分方程式木にまたがっていれば、遅延の多い方から少ない方へと遅延順序リストを作成する。連結後の木間の順序対リストを作成する。計算グラフの最も深い木から出発して遅延順序リストを考慮しながら順序対リストに従って差分方程式木の計算順序を決定して行く。図１０に、２



$$
\begin{cases}
u(n) = x_1(n) - y_2(n-1) \\
x_2(n) = x_1(n) + k_2 \cdot u(n) \\
y_1(n) = y_2(n-1) + k_2 \cdot u(n) \\
v(n) = x_2(n) - y_3(n-1) \\
x_3(n) = x_2(n) + k_1 \cdot v(n) \\
y_2(n) = y_3(n-1) + k_1 \cdot v(n) \\
y_3(n) = x_3(n)
\end{cases}
$$

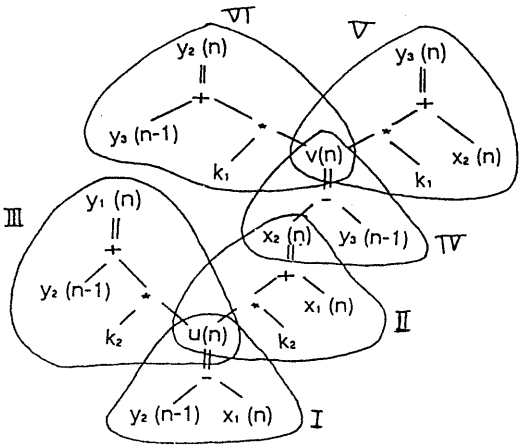
図１０：２次オールパスラチス回路

次オールパスラチス回路と連立差分方程式と、式（３），（４）に遅延順序リストと順序対リストを示す。

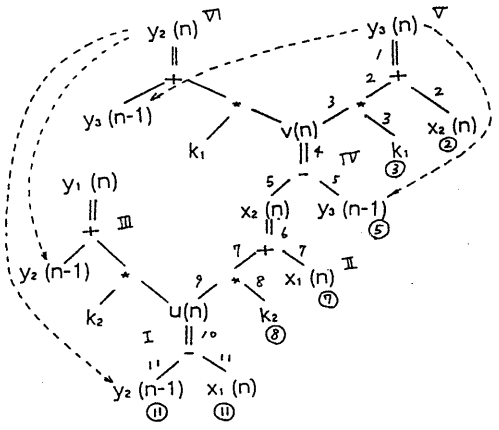
n=0　n=1　　n=0　n=1･･･遅延量

y_2:（Ⅵ，Ⅱ），（Ⅵ，Ⅰ）
y_3:（Ⅴ，Ⅵ），（Ⅴ，Ⅳ）　(3)

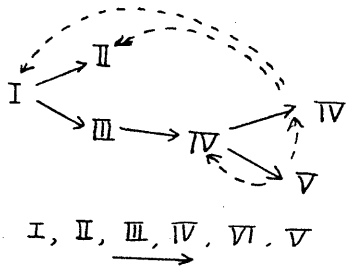
(Ⅱ, Ⅰ),(Ⅲ,Ⅰ),(Ⅳ,Ⅲ),(Ⅴ,Ⅳ),(Ⅵ,Ⅳ)　(4)



（ａ）差分方程式木の連結



（ｂ）葉の深さと遅延順序（点線）



（ｃ）木の計算順序

図１１:木の計算順序決定

即ち、計算グラフ中の木の計算順序は、

(1)木の連結:番号づけられた差分方程式木の根が、他の差分方程式木の葉に同じなら、根の方の木を他方の部分木とする。これを全ての木について行い計算グラフとする。（図１１の（ａ））

(2)葉の深さ:計算グラフの全ての葉に、代入される変数節から各葉までの深さのうち最も深い深さを記す。（図１１の（ｂ））

(3)遅延順序リスト:全計算グラフに渡って、変数ごとに遅延の少ない順に矢印を付け、矢印が木と木に渡るとき、その順で木の番号をリストに加える。

(4)順序対リスト:木の連結の際に、それぞれ連結部での親子の木の対の番号を（親、子）の順でリストへ加える。

(5)順序対による木計算順リスト:(2)での最も深い葉を持つ木から始めて、順序対リストをたどり、最上位木まで順に並べる。ことき、全ての木が上位木を２つと持たなければ、その順が木の計算順序となる。

一般には、

(6)木の計算順序:(5)での順のうち、上位木を二つ以上持つ木が有れば、より小さい上位部分木を持つ方、かつまたは、(3)での各変数の遅延順序リストのうち、より大きい遅延を持つことを示す木を選ぶ。（図１１の（ｃ））

な手順で決定する。各木のコード生成は、３．４ま

での手順に従う。木の連結節では下位部分木の評価値がアキュムレータに存在することより、使用レジスタの変更等を必要とする。

自動的な木連結により、本コード生成システムの利用者は差分方程式間の計算順序に煩わされることはない。また、下位木の評価結果を直接上位木で利用できることからロード命令の節減となる。

4．プログラム生成例

本プログラム自動生成法を、4種のディジタルフィルタ回路について適用した結果が表1である。

表1：各種フィルタのプログラム生成結果

Filter Type	Beginner	Optimized	Speed-Up Ratio
1D-NOCH	37(34)	20(11)	1.85(3.09)
2D-NOCH	44(41)	20(11)	2.20(3.73)
2次ALLPASS直接	38(35)	20(11)	1.90(3.18)
2次ALLPASSラチス	53(50)	31(22)	2.27(2.127)

表中央部の左の数は初心者によるプログラムの実行サイクルを、右の数は本方法により自動生成されたプログラムの実行サイクル数を示す。数字は総実行サイクル数、括弧内の数は演算部のみの実行サイクル数を示す。初心者のプログラムとは、ＤＳＰのアーキテクチャを無視し、単にデータロード、演算、結果格納のパターンを繰り返すプログラムである。

速度向上比は初心者のプログラムに対して、本プログラム生成法によるプログラムは、総実行サイクル数で約1/2、演算部で約1/3になっていることを示している。1D-NOCHと2次ALLPASSラチス回路では木の連結によりさらに節減される。

5　まとめ

ディジタルフィルタの差分方程式から、5C72アセンブラプログラムを自動生成する方法を述べ、実験結果を示した。プログラムの生成法は、遅延処理の最適化、データバス使用法の最適化、パイプライン密度の向上、計算木、計算グラフによるコード生成の各段階よりなっている。プログラムの生成は数種のフィルタについて行い、良好な結果を得た。

今後の課題は、複雑な構成のフィルタ回路、即ち、連立差分方程式からのプログラム自動生成法における、木の連結部付近での命令木の検索（レジスタの割当）に改良を要する。

謝辞

有益なコメントを戴いた東工大の西原教授、杉野助手、ＮＥＣの西谷博士、黒田氏、福井大学情報工学科の浅田教授、金沢工大渡邊教授に謝意を表する。

参考文献

[1] J.Hartung, S.I.Gay, and S.G.Haigh, "A practical C Language compiler/optimizer for real-time implementations on a family of floating point DSPs", Proc. of ICASSP, pp.1674-1677(April,1988)

[2] I.Kuroda, A.Hirano, and T.Nishitani, " A knowledge-based compiler enhanceing DSP internal parallelism",Proc. of IEEE ISCAS'91,Vol.1,pp.236-239(june,1991)

[3] 杉野,年清,渡辺,西原,"ディジタル信号処理回路の計算順序決定とそのシングナルプロセッサ用コンパイラの応用", 信学論(A),Vol.J71-A, No.2,pp.327-335(1988-02)

[4] 山村,清井," 計算グラフを用いた非線形関数の分離性検出アルゴリズム",信学論,Vol.J74-A, No.12,pp.1755-1765(1991-12)

[5] G.j.Chaitin, " Register allocation and spilling via graph coloring",Proc. of ACM SIGPLAN,ACM(June,1982)

[6] 北岡、加藤、石端、武部、"差分方程式によるディジタルフィルタプログラム自動生成",第7回ディジタル信号処理シンポジウム講演論文集, pp.309-314(1992-11)