C-007

LSI設計支援を指向したC言語からのCDFG作成手法

A CDFG Generating Method from C Program for LSI Design

加藤 俊之† 宮内 崇旭† 三宅 貴章† 大亦 真一† 西門 秀人† 山内 寛紀† 小林 士朗‡

Toshiyuki Kato Takaaki Miyauchi Takaaki Miyake shinichi Omata Hideto Nishikado Hironori Yamauchi Shiro Kobayashi

1. はじめに

近年，音声や画像を扱う電子機器が普及している．そしてLSIの集積化に伴い，ますます多くの機能が搭載されるようになった．これらの機能を満たすにはリアルタイム性が求められ，この処理を行うために搭載されるのがDSP(Digital Signal Processor)を代表とするメディアプロセッサである．これらのプロセッサは信号処理特有の演算を高速処理できるほか，低消費電力なためポータビリティ性にも優れている．このような近年の動向の中，LSI設計者には大きな負担がかかるようになった．これは，アプリケーションの大規模化に加えて，商品の多様化のために開発期間は短縮される傾向にあるからである．現在のこれらのLSIの開発手法は抽象度の高い言語からの高位合成が主流であり，C言語による動作記述から回路を自動生成する動作合成が開発現場で用いられる様になってきている．しかし，既存の手法[1]で信号処理LSIを開発する際は，HW処理部とSW処理部とを人が指定しなければならないため，自動化には至らずまた，専用HW化が関数単位での指定のため，効率的な専用HW化と必ずしも言えない．加えて，現在メディアプロセッサのアーキテクチャを最大限に活かすコンパイラの開発は困難なため，アセンブリによる開発環境が多く，ますます大きな負担をLSI設計者は強いられる．そこで本研究室ではC言語によって記述されたアプリケーションからメディアプロセッサのアーキテクチャ及び，実行コードを自動生成するシステムの開発を行った．これにより，設計開発期間を短縮し，設計者の負担を減らすことが可能となる．また，演算子単位で頻出する複合演算の抽出，HW化を行うため効率の良いアーキテクチャを生成できる．

以下，2で本研究室が提案するCDFGについて述べ，3でプロファイラの処理フロー，4で実装結果を示し、5で考察，6でまとめとする．

2. CDFG生成システム

2.1. CDFG

DFG(Data Flow Graph)とはデータの流れを表したグラフの事をいう．DFGの例を図1の左に示す．円で囲まれた演算子は演算処理を表し，各円を繋ぐ線はデータの流れを表している．このDFGはA=(B+C)+Dを表している．CDFGとは，このDFGに,分岐や繰り返し等の制御情報を加えたグラフの事をいう．一般的なCDFGを図3の右に示す．ここでの三角形は分岐処理を表しており，制御変数'x'の値によって異なる演算処理が行われる．このCDFGは'x'が0ならa=(b-d)+c; 'x'が1ならa=b+d; の処理が行われる事を示している．次項に本研究室の提案するCDFGの説明を行う．

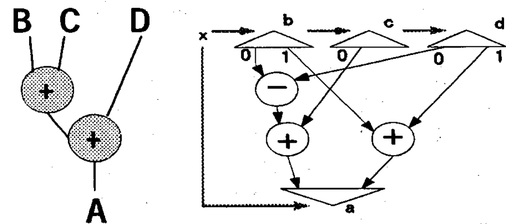

図 1 DFG(Data Flow Graph)と CDFG(Control Data Flow Graph)

2.2. 提案するCDFG

まず一般的な特徴としてはC言語プログラムをCDFGに変換することによってコーディングスタイルによる解析結果のバラつきを防ぐことができる．またCDFG自体がハードウェアに近いデータ形式なため，複合演算等のHW化が容易に行える．本研究室で提案するCDFGは上記の一般的な特徴に加え，以下の特徴を持たせることにより，アプリケーションごとに最適なメディアプロセッサの自動合成を可能にすると考える．

CDFGの階層化

Cソースプログラムの構造を維持したプログラム分割（タスク分割）及び階層的なCDFGを生成する．これにより，任意の粒度において演算量と並列度の情報を提供でき，ソフトウェア（アルゴリズム）の視点からハードウェアアーキテクチャ選択の試行錯誤が可能となる．

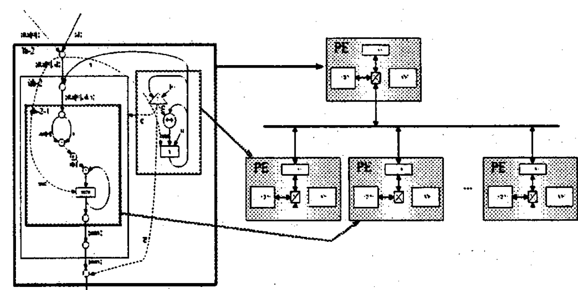

図 2 CDFGからのHWマッピング例

† 立命館大学理工学部 Ritsumeikan Uviversity
‡ 旭化成株式会社研究開発本部 Asahikasei Corporation

● データ演算，アドレス演算の区別化

配列，ポインタ等のメモリアクセスに関してはメモリアクセスノードと名づけた演算子をCDFG中に設けている．アドレス演算の際には必ずメモリアクセスノードを挟むため，データ演算，アドレス演算の区別化が可能となる．これにより本研究室が提案するVLIW型のアーキテクチャの性能を十分に活かし，並列性を高めることができる．

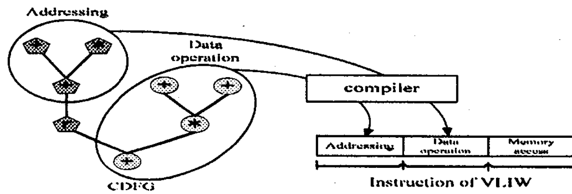

図 3 データ，アドレス演算分離による並列性

● 階層ごとの演算量・並列度の算出

階層ごとに演算量・並列度の算出ができるので，階層毎に対するHWアーキテクチャの選択が可能となる．また、ユーザーが各演算について演算量を指定することが出来るため専用演算器を持つことを想定した解析結果を得ることが可能．これにより演算器の採用に利用できる．

● 複合演算の抽出

CDFG内から複合演算の全パターンを探し出し，頻出する演算の抽出を行う．そしてユーザーの要求に応じてその複合演算の専用HWを生成し，それに対する命令セットも自動生成する．この処理は階層毎に行うため，階層毎に最適なアーキテクチャを生成することが可能となる．

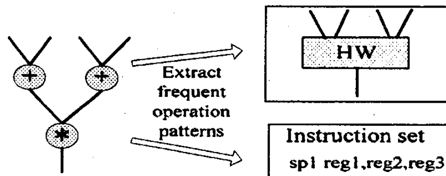

図 4 複合演算抽出，専用HW化，命令セット生成

3. 処理フロー

本稿で提案するCDFG生成処理フローを図7に示す．入力となるのはC言語記述されたアプリケーションで，出力は階層型のCDFGである。

まず入力となるC言語はプリプロセスにより，インクルード等の展開が行われる．また記述の自由度の高いため，解析が困難である構造体に対しては，制約を設けた上で処理のしやすい形式に変換を行っている．次はC言語を制御等の処理単位に分割していく．この処理単位をタスクと呼び，またこれが階層の基準ともなる．次に分割したタスク間の依存解析を行っていく．ここで一旦CDFGのデータを生成する．そしてそのデータを元にデータ演算，アドレス演算に分離を行っていく．そして分離後，ユーザーの指定した各演算子の重み付けを元に各タスクにおける演算量，並列度を求め，プロセッサの階層構造構築等の指標とする．以上の処理をもとに階層型CDFGを生成する．

次にこれらの処理について詳しく述べる．

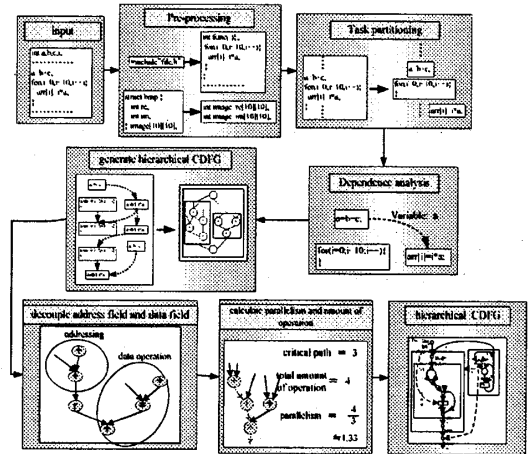

図.5 CDFG生成フロー

3.1. Pre-processing

プリプロセスにおいてはマクロで定義された変数を展開や，ヘッダファイルのインクルードを行う．また宣言の自由度が高い構造体は，このままでは解析が困難である．このため処理がしやすいように変数とメンバの数だけ展開して個別に宣言を行う．画像処理では構造体を用いた実部，虚部の表現が頻繁に利用されるため，この変換がメディアプロセッサ合成には不可欠となる．

3.2. Task partitioning

タスク分割はyacc/lexという字句解析，構文解析を行う言語を用いて行った．分割の定義は表1に表した通りで，基本的にはC言語の処理単位で分類されていく．またこれが階層型CDFGの階層の基準となる．

表 1 type of Task

Granularity	Type of Task	Definition
Fine grain level	FB (Function Block)	Total definition of function
	LB (Loop Block)	Loop statement sets
	SLB (Selection Block)	Branch statement sets
	SB (Statement Block)	Statement sets
Medium grain level	ITB (Iteration Block)	Iteration statement sets in loop block
	BRB (Branch Block)	Branch target statement block
Course grain level	ST (Statement)	Statement of program
	FS (Function Statement)	Statement including function call

3.3. dependence analysis

依存解析部では全タスク間の依存関係を求めていく．依存の種類は図6に示す．定義した変数が使用されるフロー依存，使用した変数が定義される逆依存，定義された変数が再定義される出力依存がある．

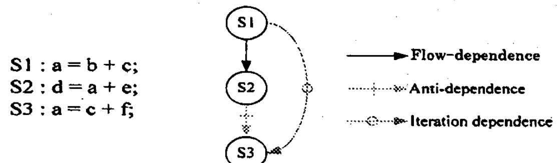

図 6 依存の種類

また実際の解析においては到達する定義[2]と呼ばれる手法を用いている（数式1）．あるステートメン

トに到達する（REACH）変数は，一つ前のステートメントにおいて，定義された（DEF）変数，もしくはそのステートメントに到達した変数からそのステートメントによって無効にされた（KILL）変数を差し引いた物，との集合となる．

$$REACH(S) = \bigcup_{P \in pred(S)} [DEF(P) \cup \{REACH(P) - KILL(P)\}] \quad (1)$$

またループ部分に関しては繰越依存の解析が依存解析部での最大の焦点となる．繰越依存とはある時点のループで定義された変数が次以降のループへ依存する事である（図 7）．繰越依存の正確な解析により，最もLSIの性能が左右されるループ部分の処理手法を決めることとなる．そして特に解析が困難となるのは，配列から配列へのオフセットの変化による繰越依存である．本提案手法では，GCD(Greatest Common Divisor)テストで繰越依存が起こりうるかの判定を行っている．この手法ではまず依存元と先の配列のオフセット部分で等式をたてる．これをディオファンタス方程式と呼ぶ．またループの制御に使われる変数（図 7 では i）をループインデックスと名づける．等式中に存在するループインデックスの係数の最大公約数（GCD）で，ディオファンタス方程式の定数部分を割り切れれば，この等式を満たすループインデックスが存在することとなる．つまり，配列間においてループをまたいでの依存が存在するという事になる．

```
for (i = 0; i < n; i++) {
    a[i]     = b[i] + c;
    b[i + 1] = a[i] + d;
}
```

Diophantine equation

$$i_1 = i_2 + 1$$
$$i_1 - i_2 = 1$$

図 7　繰越依存とディオファンタス方程式

3.4．データ演算・アドレス演算分離

データ演算，アドレス演算の分離により，VLIW 型命令の性能を引出すことができる．しかし，CDFG というのはデータの流れを表すものであり，メモリへのアクセスを示すことはできない．そこで本提案手法ではメモリアクセスノードという CDFG 上でのノードを設け，メモリへのアクセスを表現した．図１０にそのメモリアクセスノードを示す．図中の左側の R（Read Node）は「読み出し」を表しており，入力はアドレス，出力はアドレス先のデータを示す．配列の要素へのアクセスは，初期番地とオフセット分の和を入力のアドレスとすることで可能である．そして図中の右側の W(Write Node)は「書き込み」を表しており，書き込みたいデータとそのアドレスを入力する．出力はアドレス先が更新されたという情報である．配列の要素への書き込みは Read Node で説明した場合と同様初期番地とオフセット分の和がアドレスの入力値となる．

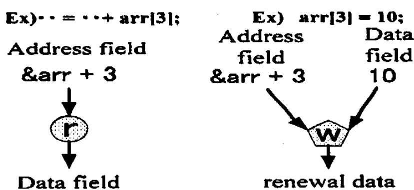

図 10　node of memory access

このメモリアクセスノードを挟むことによりデータ，アドレス演算の分離がしやすくなる．更に CDFG で全演算の結合を行っているので，オフセット部に用いられた変数が定義されたステートメントに向かって，さかのぼって解析していくことが可能となる．また，ループ内の処理において，ループを繰越して配列のオフセットとなっているケースにおいてもアドレス演算と解析することも可能とした．

3.5．演算量・並列度算出

演算量と並列度の算出はタスク単位で行っている．これらの算出例を図 11 に示す．演算量は処理時間と同義であり，各演算子の演算量（処理時間）が記載された演算量定義ファイルをもとに算出する．演算量定義ファイルは合成対象となるハードウェアアーキテクチャに合わせて自由に値を設定することができる．各タスクの演算量はそのタスクの CDFG 上に存在する各演算子の演算量の総和として算出し，関数コールを含む場合は，コールされる関数の演算量も加算して算出する．下層タスクを持つタスクは，そのタスク内に存在する全ての下層タスクの演算量の総和をそのタスクの演算量とする．演算量算出処理は最下層タスクから上層タスクへと順に行い，最終的に関数全体の演算量を算出する．なお，LB と SLB の演算量は以下の手法で算出する．LB では，ループカウンタの演算量と ITB の演算量をそれぞれ算出し，ITB の演算量にループの繰返し回数を掛けた値を LB の演算量とする．これは，信号処理プロセッサでは，ALU とは別にアドレス演算器を持つものが多く，ループカウンタの演算はアドレス演算器で行うことが多いためである．SLB の演算量は，各 BRB の演算量のうち最も演算量が多いものに条件部の演算量を加えたものとする．各 BRB のうち最も演算量が多いものを採用するのは，本合成システムが信号処理 LSI などのリアルタイム向け並列システムの合成を対象としているからであり，分岐先に依存してリアルタイム性を満たすか否かが決定されるのを防ぐためである．

また，並列度は並列処理による処理能力の向上率を表すものであり，ハードウェア合成部でのハードウェア構成決定において重要な指標となる．並列度は，タスクの演算量をそのタスクのクリティカルパス長で割ったものと定義する．ここでクリティカルパスとは，タスクの CDFG 上における始点 VN から終点 VN までに複数存在するパスのうち，最も演算量の総和が多いパスのことであり，クリティカルパス長とはクリティカルパスの演算量を表す．並列度算出は，演算量算出と同様に最下層タスクから上層タスクへと順に算出し，最終的に関数全体の並列度を算出する．

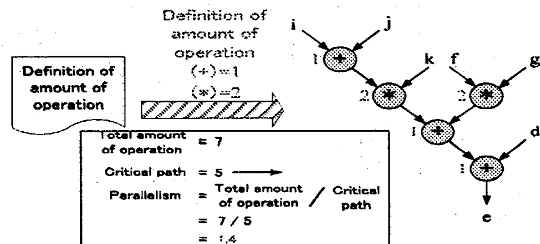

図 8　演算量・並列度算出

4. 実装結果

今回は FFT（高速フーリエ変換）で用いられるバタフライ演算の一部のプログラムを入力として実装の結果を示す．入力ソースは図 9 に示し，実装結果である階層型 CDFG を図 10 に、並列度・演算量を図１４に示す．実装結果の図に関しては上記で説明したグラフィック生成を用いることで実装の検証を行う．

図 13 では、実装結果の丸いノードがデータ演算で，四角のノードがアドレス演算である．四角のフレームが階層を示している．図 11 では、各タクス毎に総演算量(total)，データ総演算量(data)，アドレス総演算量(address)，総クリティカルパス(t_pass)，データ演算クリティカルパス(d_pass), アドレス演算クリティカルパス(a_pass)を示しており，枠線で囲ったタスクが最もコアな演算タスクとなる．但し，演算量は*=3,/=5,それ以外の演算を 1 としている．

```
for (i = 0; i < num_of_dft; i ++ ){
  re_buf = *( re_part + num_in2 );
  im_buf = *( im_part + num_in2 );
  re_part_new[num_out] = *( re_part + num_in1 )
                         + re_buf * cos[angle] + im_buf * sin[angle];
  im_part_new[num_out] = *( im_part + num_in1 )
                         + im_buf * cos[angle] - re_buf * sin[angle];
}
```

図 9 バタフライ演算（一部）

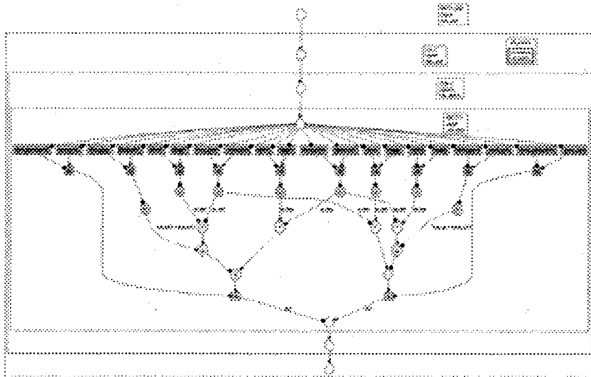

図 10 実装結果

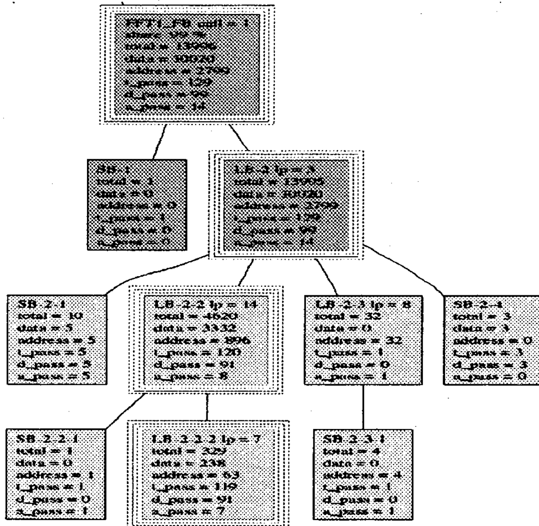

図 11 実装結果

5. 考察

図 10 からデータの依存解析，データアドレス演算の分離，演算量算出などの結果が正しいことを確認できた．今回は紙面サイズの都合上バタフライ演算の一部しか図を載せることができなかったが，実際には一次元の FFT のソースに対しても解析結果を確認している．また，図 11 のループ部分 LB-2, LB2-2, LB-2-2-2 の並列度はそれぞれ 3.257, 3.254, 3.286 となっており，FFT 全体の並列度は 3,256 なので，3 並列処理することが効率がいいことが分かる．提示した図の情報や，バイナリデータの解析情報からも本研究室で提案する CDFG がプロセッサ合成への中間データとして有効であることが確認できるとともに，実用レベルに近い入力の解析が可能であることが確認できた．

6. まとめ

近年の LSI の高性能化に伴う，LSI 設計者への負担のしわ寄せを解消するために，本研究室では C 言語からのプロセッサアーキテクチャと実行コード生成による自動合成システムを提案した．そして本稿ではその中でも C 言語を解析し，中間データである CDFG への変換手法及び特徴を述べた．

提案する CDFG は階層的に構築されることから，ハードウェアのアーキテクチャの階層の振り分けが容易となっている．またデータ演算，アドレス演算の分離から，VLIW 型命令を採用している本研究室のプロセッサの並列度を上げることも可能となる．また CDFG 情報から複合演算抽出し頻出の演算を専用岐路化することで，最適な HW/SW 処理分割が可能となる．

これらの特徴をバタフライ演算の一部を入力とした実装結果により確認することができた．

今後はループ部分の依存の解析精度の向上，現在制約となっている特殊な C 言語記述への対応に取り組んでいく．

文献

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

[2] 中田育男，" コンパイラの構成と最適化"，朝倉邦造，2002

[3] 笠原博徳" シングルマルチプロセッサ上での近細粒度並列処理" 情報処理学会論文誌 Vol.40 No.5 pp.1924-1934

[4] 西口健一" ソフトウェア互換ハードウェアを合成する高位合成システム CCAP における変数と関数の扱い" 電子情報通信学会技術研究報告. VLD2005-79(2005-12)