プログラミング言語 24－2
（1990. 2. 9）

並列記述言語 DFCII の命令レベル データ駆動計算機に対する構造文処理

関口智嗣, 島田俊夫, 平木敬

電子技術総合研究所

電子技術総合研究所では次世代スーパーコンピュータを目指して，命令レベルデータ駆動計算機 SIGMA–1 の開発を行なってきている．SIGMA–1 の高級言語とした並列処理記述用言語として C の文法を元にした DFCII の設計を行なった．これは C 言語と同様に if 文，for 文，do-while 文，switch-case 文等の構造文がある．これらの構造文を命令レベルデータ駆動計算機用の命令セットを用いて記述する際の分岐命令の決定法と同期トークンの処理法について述べる．すなわち，これまでに提案と実装を行なっている bsw 命令を用いることと，変数の定義参照関係を解析することにより適切な分岐命令の決定法を述べる．

A Decision Principle of Switch Nodes in Parallel Language DFCII

Satoshi Sekiguchi, Toshio Shimada and Kei Hiraki

Electrotechnical Laboratory
Umezono,Tsukuba,Ibaraki 305, JAPAN

The SIGMA–1 system, which is the first and the fastest instruction level dataflow computer in the world has been developed as a supercomputer. for the next generation by the Electrotechnical Laboratory, The DFCII has been designed and implemented as a high level language for the SIGMA–1 system. The syntax of the DFCII looks like the conventional C language, that is the DFCII has some compound statements as if-statement, for-statement, do-while-statement and switch-case-statement. In this manuscript, the decision principle of switch nodes when compiling compound statements in the DFCII are described.

1 はじめに

電子技術総合研究所では次世代スーパーコンピュータを目指して，命令レベルデータ駆動計算機SIGMA-1の開発を行なってきている．この命令レベルデータ駆動計算機はデータ依存グラフとして表現されたプログラムに内在する関数レベル，文レベル，変数レベルの並列性を自然な形で抽出して並列に実行するのが大きな特徴である．

また，ユーザはデータ依存グラフを直接に記述するわけではなく，高級言語を用いて実行させたいプログラムを記述するのが一般的である．SIGMA-1でも高級言語としてCの文法に類似した形式を持つDFCIIの設計を行なった[1]．DFCIIとC言語は文法的に

- goto文とループ構造文における break，continue の排除，
- 構造文相互の入れ子構造の禁止，
- 同期構造文などの導入，

という違いがある．

これまでのデータ駆動計算機に適した言語は単一代入則の導入で関数型言語に近い性質を持っていた．関数型言語は関数間や変数間の依存性解析が非常に容易になるという利点があるが，その反面，ユーザに対して単一代入則の制限を強制するといった不都合な点もある．さらに，ループ構造におけるループ世代間に跨った変数の関係のように，みかけ上，単一代入則を破らざるを得ない場合がある．それらの扱いには，new，oldといった属性の記述を行なうVal[2]，Id[3]，SISAL[4]やfor文の一部分に限定して文法的例外を認めるDFC[5]などがある．

これらに対して，DFCIIでは単一代入則を排除し，変数の依存性解析をコンパイラが行なう．ここで，単一代入則に相当する解析ルールとしてDFCIIは文並びの順に逐次解釈を行なうルールとした．この解析ルールに基づいて変数の依存性解析を行ない，DFCIIからデータ依存グラフを生成する．

DFCIIはC言語と同様にif文，for文，do-while文，switch-case文等の構造文があり，このような構造文から逐次解釈に基づくデータ依存解析によるデータ駆動計算機をターゲットとした処理系作成のガイドラインは従来の研究では与えられていない．

本稿では，DFCII処理系の概要と処理系設計の基本となるブロックの考え方について述べる．さらに，データ駆動計算機において構造文を実現するために必要な分岐命令(sw)とその改良について述べ，各種構造文におけるブロック構造とswの選択方法について述べる．

2 DFCII処理系の特徴

2.1 DFCII処理系のブロックによる設計

データ駆動計算機はデータ依存グラフと呼ばれるグラフを直接に実行する．データ依存グラフはノードとそれをつなぐアークからなる．このアーク上をデータトークンと呼ばれるデータが流れ，それぞれのノードにおいてペアートークンとの待ち合わせとノードの実行を行なう．ノードの実行が行なわれることをノードの発火と呼び，ノードが発火した後は通常，入力側のトークンは消滅し出力側のトークンとして新たにアーク上を流れていく．また，プログラムで現れる変数は一部のアークに相当する．

データ依存グラフの基本構成要素には，図1に示す5種類ある．演算(operation)ノードは単数または複数の入力トークンが全て揃った時点で演算を実行して出力を生成するノードで，加減乗除，比較などの算術演算として最も基本的なノードである．複写(copy)ノードはデータトークンのコピーを行なう．併合(merge)ノードは複数の入力アークのうちいずれか1つに限って選択的にデータトークンが流れてくることを保証するノードである．この入力アークの唯一性の論理的な保証はグラフ記述者に委ねられている．吸収(absorb)ノードは唯一の出力アークを持たないノードである．通常はプログラムの終端として用いられる．分岐(branch)ノードは制御値の値により，データの出力アークを選択するノードである．一般には制御値として真偽値をとる．

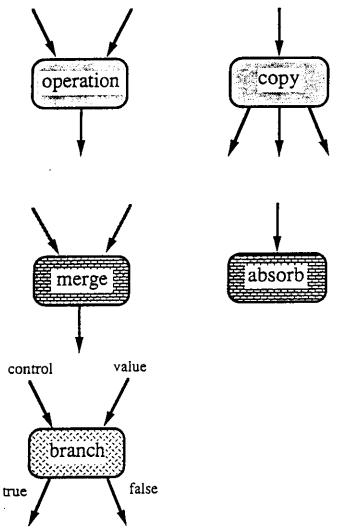

図 1: データ依存グラフの基本構成要素

SIGMA-1の場合，それぞれのノードは2つ以下の入力アークに関する待ち合わせを行い，出力アークを3本まで出すことができる．これはハードウェアの制約やアーキテクチャの設計による数値である．したがって，ハードウェアやアーキテクチャに依存しないグラフを考えるにはグラフの基本構成要素を組合せて，多入力多出力の大きなノードを考えればよい．この大きなノードのことをブロックと呼ぶ．

グラフの基本構成要素を任意の組合わせただけでは安全なブロックにならない．ここでいう安全性の必要十分条件は

1. 入力のないアークやどこにも接続されていない出力アークが存在しない，
2. どのような入力に対しても出力値が全て確定する，
3. 出力値が全て確定した場合には，ブロック内部にトークンが残留しない，

ことである．1の条件は，入力が全てブロックの外部より与えられ，ブロック内部で盲腸となった出力アークが存在

しないことである．2の条件は，分岐ノードを含むような場合でも，制御値によらず，すべての出力値が確定することを要請している．3の条件は，全ての出力値が確定したのにもかかわらず，ブロック内部のアーク上でペアーを待つトークンが残留していないことを要請している．この条件により，全ての出力値が確定した時点でブロック内部の必要なノードは全て発火したことを保証し，当該ブロックを消去することが可能となる．

DFCII処理系の設計にあたっての基本的な考え方として，このブロックによる再帰的な構成をおこなった．すなわち，DFCIIにおける文や三項式などの一部の式に対して，処理の基本単位であるブロックを適用した．このブロックでは変数の複数入力，複数出力を認め，あたかも玩具の組み立てブロックのようにブロック相互で組み合わせることでプログラムを構成する．ブロックの入出力変数が与えられる口をそれぞれ入力ポート，出力ポートと呼ぶ．DFCIIではこのブロックの考えを取り入れた中間言語 SASGA の設計も併せて行なった[7]．

さて，ブロックの中身は単純な代入文の集まりについては演算ノードの組合せとして，安全なブロックは容易に作成できるが，分岐構造やループ構造を持った文において安全なブロックの作成は単純ではない．分岐構造文やループ構造文の処理法において，SASGA では文に対応したブロックの内部にいくつかの機能ブロックを設定し，安全なブロックの作成を行なった．

2.2 DFCIIの解釈ルール

従来，データ駆動計算機用言語とされてきた Val，Id，SISAL といった言語は関数型言語を基にして，単一代入則を導入することで，変数名を一意にした．このため，言語処理系にとって関数間や変数間の依存性解析が非常に容易になるという利点があるが，その反面，ユーザは単一代入則に基づいて記述する必要があるため，科学技術計算などで現れるプログラムですら記述が不自由になる点も指摘されている．

これらに対して，DFCIIでは単一代入則を排除し，変数の依存性解析をコンパイラが行なう．ここで，単一代入則に相当する解析ルールとして DFCIIは C 言語と同様に文の並びにしたがって逐次解釈を行なうというルールがある．この解析ルールに基づいて変数の依存性解析を行ない，DFCIIからデータ依存グラフを生成する．たとえば，

$$
\begin{aligned}
x &= 3; \\
a &= x+1; \\
b &= a; \\
a &= x+2; \\
c &= a;
\end{aligned}
$$

という場合，単一代入則を導入した言語では a の再代入でエラーとなるが，DFCIIでは b には 4 が代入され，cには 5 が代入されるようになる．すなわち，同一名の変数に代入を行なった場合にはそれ以降の変数参照では後での代入が有効な値となる．これは，従来の C 言語と同様な結果をもたらすことを保証している．

このような代入文の並びの例では，構文解析により変数定義の有効範囲が容易に解析できる．しかし，DFCIIでは従来の C 言語と同様に各種の構造文があり，その解析が必要である．構造文とは，if 文，switch-case 文のような分岐構造文，for 文，while 文，do-while 文のようなループ構造文である．DFCIIにおいてはループ構造文の変種としてループを繰り返す毎に同期をとるという同期ループ構造文として syncfor 文，syncdo-while 文が用意されている．これらの構造文の場合，分岐条件やループ条件によって変数の定義参照関係が動的に変化する．それにもかかわらず，どのような条件下においても安全なブロックが生成されることが処理系に求められている．

2.3 BSW 分岐命令

SIGMA-1 を含めた命令レベルデータ駆動計算機においてはデータ流の方向を制御するためのいわゆる分岐ノードが命令として用意されている．もちろん，構造文のセマンティクスをデータ依存グラフで表現するには分岐ノードは不可欠である．

SIGMA-1 を含めた，これまでのデータ駆動計算機では分岐ノードとして t:f = **SW**(control,value) という SW 命令形式をとっていた．すなわち，真偽値 (control) と被制御データ (value) を入力とし，control が真の場合は t へ，偽の場合は f へ value が出力される．この SW 命令では，条件の真偽にかかわらず被制御データが出力として現れるため，implicit token [5] と呼ばれる不要なデータを回収する処理が必要となる．また，この回収する処理は implicit token の有無や発生する条件により多くの場合分けが必要となりコード生成が複雑となる．

これまでに SW 命令を改良して，C 言語に現れる構造文をデータ依存グラフとして表現するのに適した分岐命令を提案をおこなった[6]．新たな分岐命令を BSW，TSW，FSW という名前で呼び，それぞれ，boolean-switch，true-switch，

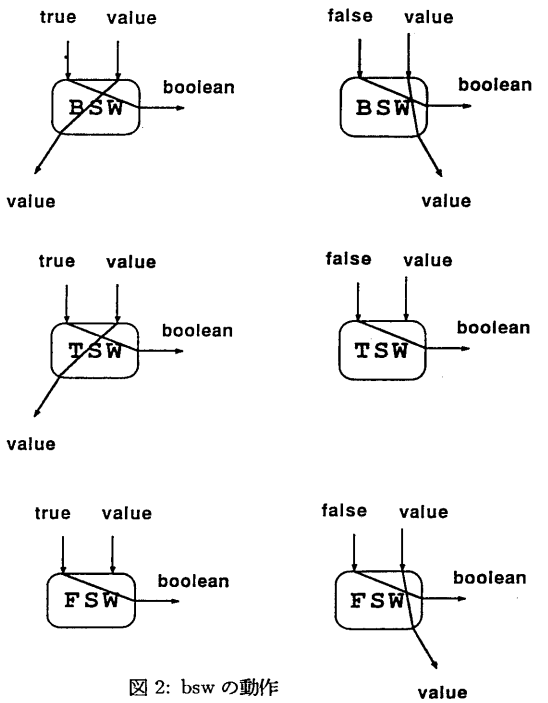

図 2: bsw の動作

false-switch の省略形を意味させる．以下ではこれらの命令の総称としてBSW 命令と呼ぶ．また，BSW 命令に関しての記述で，特に断らない限り FSW 命令，TSW 命令に関する記述を含む．この BSW 命令の形式は

control : t : f = BSW(control,value)
control : t = TSW(control,value) である．
control : f = FSW(control,value)

入力は SW 命令と同様に真偽値 (control)，被制御データ (value) である．いずれの分岐命令も真偽値の値にかかわらず，第一出力として真偽値をそのまま出力する．真偽値が真の場合に FSW，真偽値が偽の場合に TSW は第二出力のデータ値を出力しない．すなわち，TSW，FSW は真偽値の値により出力が 1 個になる．図2に動作の様子を示す．

BSW 命令は SW 命令の拡張である．SW 命令では条件にかかわらず，いずれかにデータ値を出力していたが，BSW 命令ではデータ値が不要な場合には出力を行なわない点が大きな特徴である．安全なブロックのために，何も出力しないノードの存在が認められていないため，データ値の代わりに真偽値を常に出力することで安全性を確保した．

3 分岐構造文の処理

3.1 if 文の処理

if 文は **if** (P) Q **else** R という形式をとる．Pは条件節であり，Qは then 節，Rは else 節に相当する文である．if 文の動作は，条件節の値により then 節または else 節が選択的に実行される．if 文を機能ブロックで記述すると図3のように条件節ブロック，then 節ブロック，else 節ブロック，SW 並びブロック，マージブロックとなる．

条件節ブロックは条件演算部ではPの式の値を計算し，

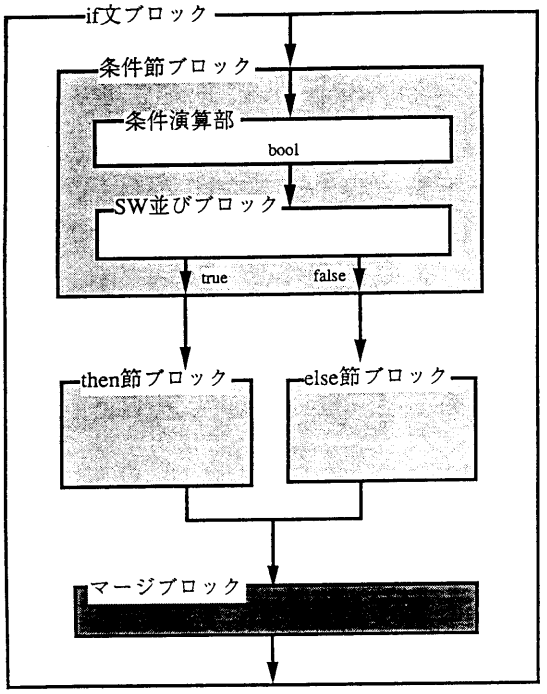

図 3: if 文ブロックの構成

論理値を定める．さらに，SW 並びブロックでは BSW，TSW，FSW をそれぞれの変数に対応するアーク上に配置し条件演算部で得られた条件値に応じてデータの行き先を制御する．then 節ブロック，else 節ブロックはそれぞれの文を実行するための文ブロックである．マージブロックは if 文ブロックが安全なブロックとなるよう外部入力，条件節，then 節，else 節における変数の値定義を併合し，出力ポートに接続する．これにより，if 文以降での参照は if 文ブロックの出力ポートを参照すれば十分となる．

さて，if 文において与える BSW 命令の種類は変数の性質として

1. 条件演算部または if 文以前に変数への値定義がある，
2. then 節での変数参照がある，
3. then 節で変数の値定義がある，
4. else 節での変数参照がある，
5. else 節で変数の値定義がある，
6. if 文以降で変数を参照している，

のうちいずれを満たしているかの組合せによって変数に対応した BSW 命令の種類とマージ (併合) ブロックへの接続の有無が決定される．

まず，当該 if 文以前に変数への値定義がある場合を考える．先の条件による BSW 命令の選択と併合ブロックへの接続を表1にまとめる．条件節を含めた if 文以前で変数の定義がない場合について考える．表2の場合以外には変数の値が未定義になる場合があるので正しいデータ依存グラフは作成できない．したがって処理系でエラーとして検出する．

なお，if 文に関して作成した決定表は C 言語における三項式 (? : 式) でも適用可能である．ただし，この場合は if 文の then 節，else 節に相当するコロンの前後は文でなく式となる．

3.2 switch–case 文の処理

switch-case 文は C 言語における形式と case ラベル，default ラベルを他の構造文の中途に設定できない点を除いてほぼ同じである．

switch-case 文は，

switch(P){
  case lab_0:
    Q_0;
    ⋮
  break;
  default:
    $Q_{default}$;
  break;
}

が一般的な形である．

switch-case 文のブロックの概略を図4に示す．switch-case ブロックはPを評価する条件式ブロック，その条件式とラベルlab_0 の同一性をテストする case 評価式ブロック，Q_0を実行する case ブロックの他，default として実行するかをテストする default 評価式ブロック，$Q_{default}$を実行する default ブロック，全体を安全なブロックにするためのマージブロックなどの機能ブロックから構成されている．

caseブロックにはさらにSW並びブロック，文並びブロック，breakブロックから構成されている．SW並びブロックはcaseブロックへ入力されたcase評価式ブロックからの真偽値により，文並びブロック以下へデータトークンを流すかどうかの制御を行なう．breakブロックは入力ポートと出力ポートを直接つなぐだけで実効を伴わないブロックであり，switch-caseブロックのマージブロックに接続される．逆にbreak文がない場合にはcaseブロックのマージブロックとして有効となる．default評価式ブロックは全てのcase評価式ブロックからの真偽値を入力とし，それらのNORを計算して評価式ブロックの真偽値とする．

ラベルとbreakの設定に関していくつかの例外が認められている．例えば，

then節		else節		外部	sw	マージ
参照	定義	参照	定義	参照		
					—	
◎					TSW	
	◎				—	
◎	◎				TSW	
		◎			FSW	
◎		◎			BSW	
	◎	◎			FSW	
◎	◎	◎			BSW	
			◎		—	
◎			◎		TSW	
	◎		◎		—	
◎	◎		◎		TSW	
		◎	◎		FSW	
◎		◎	◎		BSW	
	◎	◎	◎		FSW	
◎	◎	◎	◎		BSW	
				◎	—	
◎				◎	TSW	
	◎			◎	FSW	◎
◎	◎			◎	BSW	◎
		◎		◎	FSW	
◎		◎		◎	BSW	
	◎	◎		◎	FSW	◎
◎	◎	◎		◎	BSW	◎
			◎	◎	TSW	◎
◎			◎	◎	TSW	◎
	◎		◎	◎	—	◎
◎	◎		◎	◎	TSW	◎
		◎	◎	◎	BSW	◎
◎		◎	◎	◎	BSW	◎
	◎	◎	◎	◎	FSW	◎
◎	◎	◎	◎	◎	BSW	◎

表 1: if文における switch の決定表 (1)

then節		else節		外部	sw	マージ
参照	定義	参照	定義	参照		
	◎		◎		—	
	◎		◎	◎	—	◎

表 2: if文における switch の決定表 (2)

1. caseブロックにbreakがない場合
2. ひとつのcaseブロックに対して複数のcaseラベルがある場合
3. ひとつのcaseブロックに対してcaseラベルとdefaultラベルが張られている場合
4. 全体でdefaultラベルしかない場合

などがある．

1の場合はいずれかに後続するcaseラベルを無視して次のbreakまたはswitch-case文の終了までをひとつの大きなブロックと考える．すなわち，SW並びブロックでは外部で定義のある変数でかつ次のbreakまでに参照される変数について BSW命令を選択する．当該caseブロックで値の再定義が行なわれた変数に関してはマージポートを用意する．後続するcaseブロックでは入力ポートをすべて先行するcaseブロックの出力ポートと接続する．このラベルの設定された文から次のbreakまでをひとつのブロックと考えて同様な操作を行ない，後続するcaseブロックがないブロックまで繰り返す．

2の場合はそれぞれのラベル値と同一性を確認した後，真偽値を論理和で合成し，そのcase評価式ブロックの真偽値とする．3の場合はdefault評価式ブロックに，defaultと同じ場所に設定されたcaseラベルでのcase評価式ブロックの真偽値出力を入力し，論理和で合成する．4の場合はdeafult評価式ブロックで真値定数を発生する．

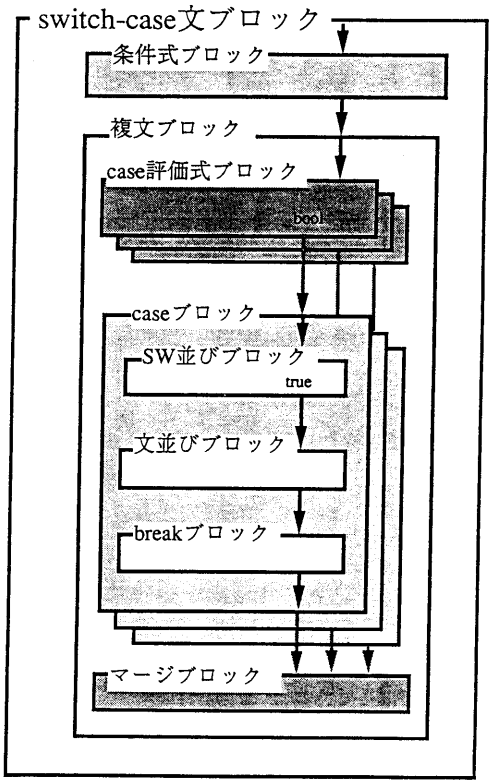

図 4: switch-case 文ブロックの構成

switch-case 文における BSW の選び方を表3，表4に示す．各変数は switch-case 文以前と条件式ブロック値定義があるものとする．そうでない場合は，安全なブロックとするためにすべての case ブロックで値が定義される必要がある．このため，switch-case ブロックのマージブロックでは全ての場合に値定義があるかどうかを判定する必要がある．

break がある場合に他所定義という条件を設定したが，これは switch-case 文の少なくともひとつ以上の case ブロックで値が再代入された変数のことである．一般には switch-case 文より先行して値が確定した変数についてのみ考えればよい．この変数を switch-case 文の外部から参照した場合にマージブロックが必要となる．このマージブロックの条件として，いかなる値に対してもマージブロックの入力アークは唯一選択されなければならないことが要請されるからである．

case ブロック		外部	break あり	
参照	定義	参照	sw	他所定義
◎			TSW	TSW
	◎		—	—
◎	◎		TSW	TSW
		◎	—	TSW
◎		◎	TSW	TSW
	◎	◎	—	—
◎	◎	◎	TSW	TSW

表 3: switch-case 文における switch の決定表 (1)

case ブロック		後続	break なし	
参照	定義	参照	sw	マージ
◎			TSW	
	◎		—	
◎	◎		TSW	
		◎	FSW	
◎		◎	BSW	
	◎	◎	FSW	◎
◎	◎	◎	BSW	◎

表 4: switch-case 文における switch の決定表 (2)

4 ループ構造文の処理

4.1 ループ不変変数

ループ構造文の実行を効率的に行なうため，ループ内部では値の更新がないループ不変変数に対して SIGMA-1 では sticky という概念を与え，毎ループでのトークン生成を抑制するハードウェア属性を導入した[8]．この sticky 属性を持つデータトークンを sticky トークンと呼び，付与されたノードにおいて対応するペアトークンが到着することにより効力を持つ．sticky トークンの付与されたノードがあたかも 1 入力で発火するノードのように振舞うためトークンの待ち合わせオーバーヘッドが消滅する．

DFC では，ループ不変変数を参照しているノードに直接付与した．すなわち，aをループ不変変数としたときにループ構造文の繰り返し部分の $x = a + y$ において加算演算ノードに sticky 属性を持つaが付与された．

しかし，直接付与方式の問題点は，yがループ不変変数である場合または $x = a + 1$ のように相手が定数の場合に，sticky トークンのペアとなるトークンがループ毎に発生しないため，ノードが発火しなくなる．このような，sticky - sticky や sticky - constant の問題は，値の先行評価と変数の依存性解析により解決することが可能であるが，処理系にかかる負担は増加する．また，同一の sticky 属性が多くのノードで必要な場合には付与と削除にかかる実行オーバーヘッドが無視できない．さらには，sticky トークンを付与，削除するタイミング，多重ループにおける処理などを実装すると安全なデータ依存グラフを生成するために言語処理系の負担がますます増加する．

DFCII では，sticky 変数を演算ノードに付与するのではなく，ループ構造文における sw 並びの switch に付与することで先にあげた問題を解決した．この場合のペアトークンは条件節ブロックからの真偽値となる．ループ構造文における機能ブロックの詳細は次節で述べる．

4.2 for 文の処理

ループ構造文として，for 文の処理を考える．DFCII は while 文もあるが，for 文の特殊な例であるので，ここでは言及しなくとも容易に類推できる．

さて，for 文は **for**(式 1; 式 2; 式 3) 文 ； という形式をとる．これに対応して，for 文のブロック構造は図5に示すように式 1 が初期化ブロック，式 2 が条件節ブロック，式 3 が増分節ブロック，文が本体ブロックとなる．さらに，条件

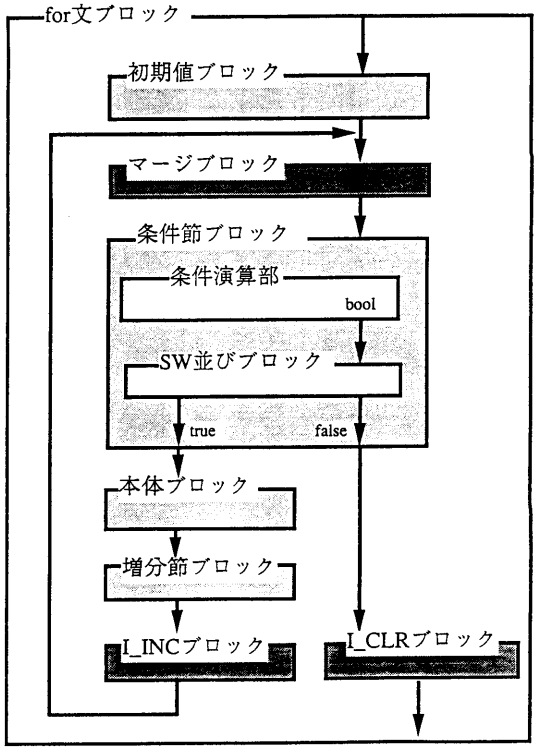

図 5: for 文ブロックの構成

節ブロックの中にSW並びブロックがあり，条件演算部で得られた真偽値によりループの継続か否かの制御を行なう．この他に，データトークンにループ識別子をつけて，どのループ世代のデータトークンであるか確定するための I_INC ブロック，ループ識別子をリセットするための I_CLR ブロックがある．

それぞれの変数についてSW並びブロックにおけるBSWの選択表を表5と表6に示す．for文では条件節ブロックの扱いと本体ブロックの扱いが異なる．すなわち，条件節ブロックは本体ブロックが実行されなくとも必ず1度は実行される．したがって，変数に関しての条件として，条件節ブロックにおける値定義と参照，本体ブロックにおける値定義と参照，さらに外部からの参照によって分類を行なった．

4.3 その他のループ構造文の処理

for文のブロックがループ構造文の基本となるが，DFCIIではその他に while文，do-while文，syncfor文，syncdo-while文がある．

while文ブロックはfor文ブロックから初期値ブロック，増分節ブロックを除いたブロックで，BSWの選択表はfor文と同じである．

do-while文の処理もfor文の処理と同様に考える．すなわち，while文ブロックの条件節ブロックと本体ブロックの位置関係が交換となる．このことは，一回目のループはループと考えず，通常の文並びとし，二回目以降からwhileループとして扱うことである．

このため，BSWを決定する表もfor文と同じであり，ループ不変変数は二回目よりSW並びに付与される

以上述べてきたfor文，while文，do-while文ではデータにループ識別子を付加することにより，ループを跨った計算の実行を可能とした．したがって，ハードウェアとしてループ識別子がタグに含まれるデータ駆動計算機ではそのままに実行が可能である．一方，ループは同時にひとつしか実行されないという制約をつけることにより，ループ識別子という計算資源を用いることなくループの実行ができる．このようなループを記述するためにDFCIIではsyncfor文，syncdo-while文が提供してある．これらの文ブロックの構造はループ識別子の操作が不要となるためfor文，do-while文とはI_INCブロック，I_CLRブロックを除いて同等である．しかし，同時にひとつのループ世代しか実行できないようにするため，I_INCブロックの代わりに同期ブロックを置く．同期ブロックは，次のループ世代へ渡される変数を串差し状にして同期をとり，全ての変数が揃った後に次のループ世代へデータトークンが渡される構造である．

5 おわりに

命令レベルデータ駆動計算機における，逐次解釈に基づく言語の特に分岐を伴う構造文の処理について述べてきた．従来，データ駆動計算機用言語として設計された言語では単一代入則を導入しており，データ依存グラフの生成が容易であった．ここで取り上げたDFCIIはC言語の文法からgotoを除いて踏襲した言語で，C言語と同様の逐次解釈を処理系が行なうことにより，単一代入則を排除することが可能となった．ここでは処理系が解釈すべき作業のうち，

条件節		本体部		外部	switch	sticky
参照	定義	参照	定義	参照		
◎					TSW	◎
	◎				—	
◎	◎				TSW	
		◎			TSW	◎
◎		◎			TSW	◎
	◎	◎			TSW	
◎	◎	◎			TSW	
			◎		—	
◎			◎		—	
	◎		◎		—	
◎	◎		◎		—	
		◎	◎		TSW	
◎		◎	◎		TSW	
	◎	◎	◎		TSW	
◎	◎	◎	◎		TSW	
				◎	—	
◎				◎	TSW	◎
	◎			◎	FSW	
◎	◎			◎	BSW	
		◎		◎	TSW	◎
◎		◎		◎	TSW	◎
	◎	◎		◎	BSW	
◎	◎	◎		◎	BSW	
			◎	◎	FSW	
◎			◎	◎	FSW	
	◎		◎	◎	FSW	
◎	◎		◎	◎	FSW	
		◎	◎	◎	BSW	
◎		◎	◎	◎	BSW	
	◎	◎	◎	◎	BSW	
◎	◎	◎	◎	◎	BSW	

表 5: for文における switch の決定表 (1)

条件節		本体部		外部	switch	sticky
参照	定義	参照	定義	参照		
	◎				—	
	◎	◎			TSW	
			◎		—	
	◎		◎		—	
	◎	◎	◎		TSW	
	◎			◎	FSW	
	◎	◎		◎	BSW	
	◎		◎	◎	FSW	
	◎	◎	◎	◎	BSW	

表 6: for文における switch の決定表 (2)

変数の定義参照関係に基づいた構造文の処理方法について述べた.

DFCII処理系は中間言語 SASGA に変換するが, この SASGA はデータ駆動計算機用中間言語としての汎用性を保っている. したがって, ここで議論した結果は逐次解釈に基づく言語から命令レベルデータ駆動計算機の命令セットに変換する際に適用可能である.

本稿では詳細に触れなかったが implicit token の構造文における処理方法, 配列構造体をアクセスした場合に生じる implicit token の吸収法, sticky token の解除のタイミング, 配列の初期化と削除のタイミングなどが処理系を設計するにあたっての課題であった. 今後はデータ駆動計算機用言語処理系における最適化の方法論について検討を行なっていきたい.

本研究は通産省大型プロジェクト「科学技術用高速計算システムの研究」一環である. 日頃より御指導頂く棟上昭男電子技術総合研究所情報アーキテクチャ部長, 田村浩一郎情報科学部長および計算機方式研究室同僚諸氏に感謝致します.

参考文献

[1] 関口 他: "同期構造を埋め込んだ SIGMA-1 用高級言語 DFCII" 情報処理学会論文誌, vol.30, pp.1639-1645, (1989).

[2] W.B.Ackerman and J.B.Dennis: "VAL-A Value Oriented Algorithmic Language : Preliminary Reference Manual," TR218,LCS, MIT,(1979).

[3] Arvind, K.P.Gostelow and W.Ploufe: "An Asynchronous Programming Language and Computing Machine," TR114a,Dept. Comp.Sci., Univ.Calif.,Irvine, (1978).

[4] J.McGraw: "SISAL: Streams and Iteration in a Single Assignment language," Language Reference Manual, LLNL, (1985).

[5] 島田 他: "データフロー言語 DFC の設計と実現" 電子情報通信学会論文誌,vol.J71-D,pp.501-508(1987).

[6] 関口 他: "命令レベルデータ駆動計算機における効率的な分岐命令の設計" 情報処理学会第 39 回全国大会, 6W-1, (1989).

[7] 関口 他: "抽象命令セットを用いたデータ駆動計算機用中間言語 SASGA" 電子情報通信学会技術研究報告, CPSY 89-36, pp.31-36, (1989).

[8] 島田 他: "科学技術計算用データ駆動計算機 SIGMA-1 のループインバリアントの処理法について" 情報処理学会第 28 回全国大会, 4F-1, (1984).