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では次世代スーパ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．

1. はじめに

電子技術総合研究所では次世代スーパ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 の開発を行ってきている[1]．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 の設計を行った[2]．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，
- 構造文内部への飛び込みの禁止，
- 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う Val[6], Id[7], SISAL[8], recur 変数として繰り返しを明示する Valid[5] や for 文の一部分に限定して文法的例外を認める DFC[3] 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う．ここで，単一代入則に相当する解析ルールとして DECII は文並びの順に逐次解釈を行うルールとした．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 文等の構造文がある．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実行に必要な相手のトークン（ペアートークン）との待ち合わせとノードの実行を行う．ノードの実行が行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

† A Decision Principle of Switch Nodes in Parallel Language DFCII by SATOSHI SEKIGUCHI, TOSHIO SHIMADA and KEI HIRAKI (Electrotechnical Laboratory).

†† 電子技術総合研究所

た，プログラムで現れる変数は一部のアークに対応する．

データ駆動計算モデルは一般に無限の計算資源が前提とされている．しかし，現実のデータ駆動計算機ではハードウェア資源が有限であるため計算資源も有限となる．たとえば，静的なデータ駆動計算モデルではグラフをコピーすることにより計算の論理的な空間の区別を行うが，ハードウェアのメモリ容量の制限により再帰的呼び出しを何度も繰り返す関数などで計算資源が枯渇することは容易に想像できる．また，動的なデータ駆動計算モデルではトークンに色をつけて区別をするため，色を表現するビット数という計算資源が不足する可能性も存在する．

このように計算資源は不明となった時点で回収を行い，何度も再利用を行うことが必要である．計算モデルを議論するだけでは問題とならなかった資源の回収という処理が実際のデータ駆動計算機においては不可欠となり，関数やあるコードブロックなどの終了を検知することがデータ駆動計算機用言語の処理系に求められている大きな特徴である．この終了検出が可能なグラフのことを安全なグラフと呼ぶ．

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すべ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．

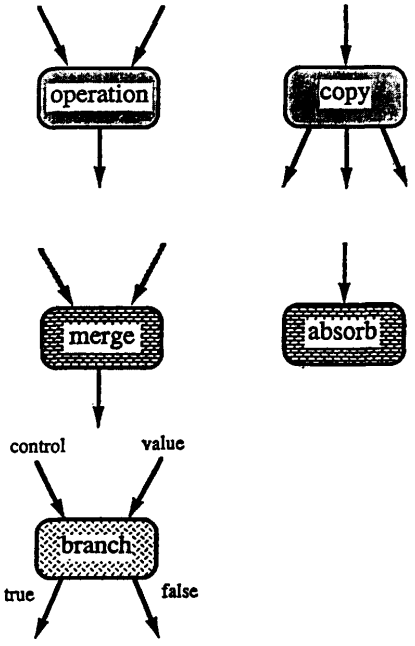


図1 データ依存グラフの基本構成要素
Fig. 1 Primitives of dataflow graph.

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み合わせて，多入力多出力の大きなノードを考えればよい．この大きな複合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に組み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すべて確定する，
3. 出力値がすべて確定しない場合には，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，入力がすべ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，すべ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，すべ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すべ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行った．すなわち，DFCII における文や式に対して，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を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 の設計も併せて行った[4]．

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い，DFCII からデータ依存グラフを生成する．たとえば，

$x=3;$
$a=x+1;$
$b=a;$
$a=x+2;$
$c=a;$

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ごとに同期をとるという同期ループ構造文として sync-for 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．

3. 効率的な分岐命令

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．したがって，真の場合に限って被制御データを参照する場合には，偽の場合に得られた被制御データは計算に用いられることなく無駄なデータトークンとして回収を行う必要がある．この計算に用いられることのないトークンを implicit token[3] と呼ぶ．

ここでは 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の提案を行う．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, 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 命令の形式は以下のとおりである．

control : t : f =BSW (control, value)
control : t =TSW (control, value)
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，真似値が偽の場

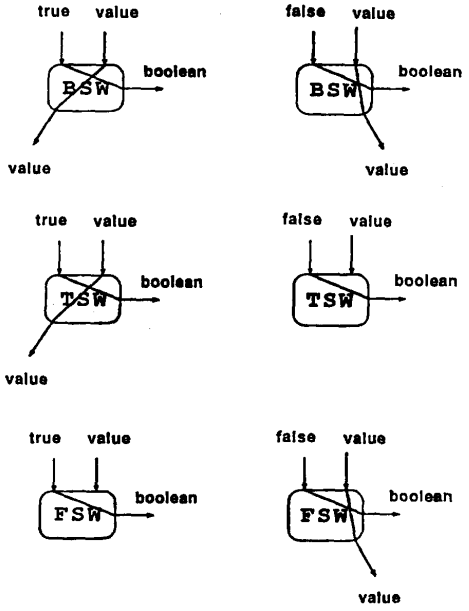


図 2 BSW の動作
Fig. 2 BSW operations.

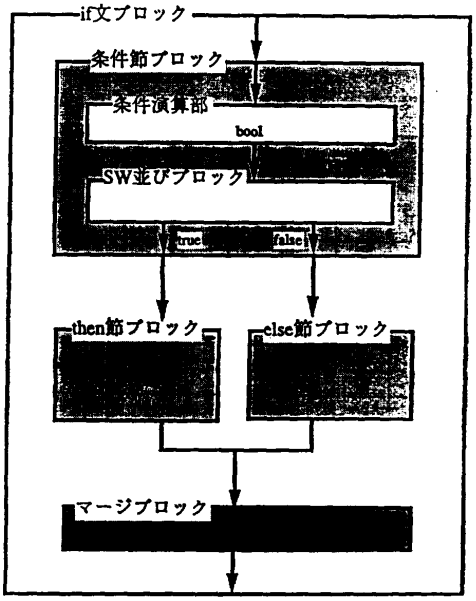


図 3 if 文ブロックの構成
Fig. 3 Component of *if* block.

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，いずれかに被制御データを出力していたが，BSW 命令では被制御データが不要な場合には出力を行わない点が大きな特徴である．しかし，特定の条件の場合に何も出力を行わないノードは吸収ノードとなるため，安全なブロックとなるための条件を満たさない．したがって，BSW 命令ではどのような条件においても真偽値を常に出力し，安全なブロックとなるための必要条件を維持した．

4. 分岐構造文の処理

4.1 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，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，

のうちいずれの条件を満たしているかの組合せに従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**に含まれない場合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．

表 1 if 文における switch の決定表（外部定義のある場合）
Table 1 Switch selection table in an *if* statement (variable defined outside).

then 節 参照	then 節 定義	else 節 参照	else 節 定義	外部 参照	sw 種類	マージ 有無
					—	
○					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 2 if 文における switch の決定表（外部定義のない場合）
Table 2 Switch selection table in an *if* statement (variable defined inside).

then 節 参照	then 節 定義	else 節 参照	else 節 定義	外部 参照	sw 種類	マージ 有無
	○		○		—	
	○		◎	○	—	◎

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 DFCII 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．

4.2 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．

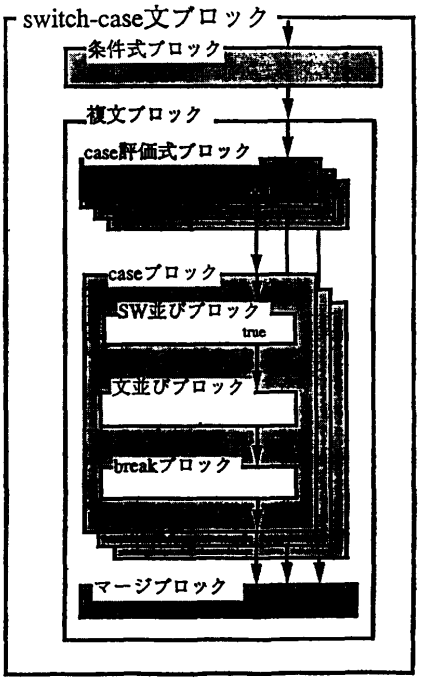


図 4 switch-case 文ブロックの構成
Fig. 4 Component of *switch-case* block.

switch-case 文は，

```
switch (P) {
        case lab0 :
                Q0 ;
                 ⋮
        break ;
        default :
                Qdefault ;
        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 ブロックのほか，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

表 3 switch-case 文における switch の決定表 (break のある場合)
Table 3 Switch selection table in a *switch-case* statement with *break*.

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

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 評価式ブロックはすべての 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．

switch-case 文における BSW の選び方を**表 3** 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すべ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 ブロックで値で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．

4.3 switch-case 文の特殊な例

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．たとえば，

1. 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
2. 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
3. 全体で default ラベルしかない場合
4. case ブロックに break がない場合

などがある．

1 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．2 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．3 の場合は default 評価式ブロックで真値定数を発生する．

4 の場合についての処理方法を示す．

case lab_0 :
　　Q_0 ;
case lab_1 :
　　Q_1 ;
case lab_2 :
　　Q_2 ;
break ;

というプログラムにおいて，まず lab_0 について後続する 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，lab_0 に関しては Q_0, Q_1, Q_2 というブロックで考え，lab_0 に対応する case ブロックの SW 並びブロックに現れる変数を定める．同様に，lab_1 については，Q_1, Q_2 のブロックを考えて，lab_1 に対応する case ブロックの SW 並びブロックに現れる変数を定める．lab_2 の場合には break があるので通常の場合と同様に処理を行う．

次に，Q_0, Q_1, Q_2 のそれぞれの文ブロックを解析し，入力ポートを決定する．Q_2 の入力変数を Q_1 の出力ポートに後続参照として登録する．Q_0 に関しても

表 4 switch-case 文における switch の決定表 (break のない場合)
Table 4 Switch selection table in a *switch-case* statement without *break*.

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

同様の処理を行う．Q_1 や Q_2 だけで参照される変数もすべて lab_0 に対応する SW 並びを経ることに注意する．

break がない場合の BSW の選び方を**表 4** に示す．

5. ループ構造文の処理

5.1 ループ不変変数

ループ構造文の実行を効率的に行うため，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[9]．この sticky 属性を持つデータトークンを sticky トークンと呼び，あるノードに対して付与される．対応するペアートークンが到着することによりノードが発火するのは通常と同じであるが，発火したあとも stickyトークンは消滅せずそのノードに残留する．このため，本来ならば2つのトークンが到着しないと発火しないノードであっても，sticky トークンは既に到着したままであるので，ペアートークンが1個到着することによりノードが発火する．したがって，トークンの待ち合わせオーバヘッドが消滅する[10]．

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，先の例における 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 トークンのペアーとなるトークンがループごとに発生しないため，ノードが発火しなくなる．すなわち，ノードに入力するべき双方のトークンが共に sticky であったり，一方が定数である場合にはノードに必要なトークンが到着しているため常に発火しているような状況になる．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．

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．すなわち，switch は演算ノードと異なり，条件節からの真偽値がループの繰り返しごとに入力され，ノードの発火をもたらすからである．

5.2 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う．このほかに，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一度は実行される．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った．

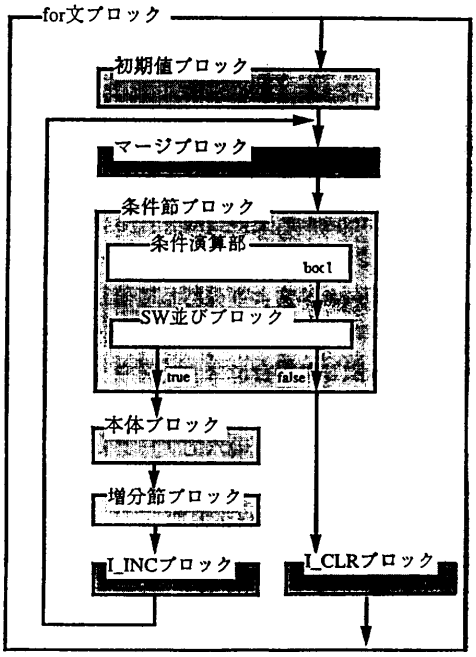


図 5 for 文ブロックの構成
Fig. 5 Component of *for* block.

5.3 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 ではそのほかに 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，すべ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．

表 5 for 文における switch の決定表（外部定義のある場合）
Table 5 Switch selection table in a *for* statement (variable defined outside).

条件節		本体部		外部	switch	sticky
参照	定義	参照	定義	参照	種類	変数
○					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 6 for 文における switch の決定表（外部定義のない場合）
Table 6 Switch selection table in a *for* statement (variable defined inside).

条件節		本体部		外部	switch	sticky
参照	定義	参照	定義	参照	種類	変数
	○				—	
	○	○			TSW	
			○		—	
	○		○		—	
	○	○	○		TSW	
	○			○	FSW	
	○	○		○	BSW	
	○		○	○	FSW	
	○	○	○	○	BSW	

6. 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，変数の定義参照関係に基づいた構造文の処理文について述べた．

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．

謝辞 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」の一環である．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いたします．

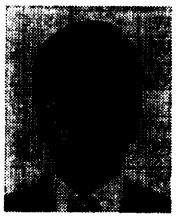
参考文献

1) Hiraki, K. et al.: The SIGMA-1 Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems, *J. Inf. Process.*, Vol. 10, No. 4, pp. 219-226 (1987).
2) 関口，島田，平木：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, No. 12, pp. 1639-1645 (1989).
3) 島田，関口，平木：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).
4) 関口，島田，平木：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).
5) 長谷川，雨宮：データフローマシン用関数型高級言語 Valid，電子情報通信学会論文誌，Vol. J71-D, pp. 1532-1539 (1988).
6) Ackerman, W. B. and Dennis, J. B.: VAL—A Value Oriented Algorithmic Language: Preliminary Reference Manual, TR 218, LCS, MIT (1979).
7) Arvind, Gostelow, K. P. and Ploufe, W.: An Asynchronous Programming Language and Computing Machine, TR 114a, Dept. Comp. Sci., Univ. Calif., Irvine (1978).
8) McGraw, J.: SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual, LLNL (1985).
9) 島田，平木，西田：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，第 28 回情報処理学会全国大会論文集，pp. 135-136 (1984).
10) Shimada, T., Hiraki, K., Nishida, K. and Sekiguchi, S.: Evaluation of a Prototype Dataflow Processor of the SIGMA-1 for Scientific Computations, *Proc. 13th Ann. Int. Symp. on Comput. Arch.*, pp. 226-234 (1986).

(平成2年2月28日受付)
(平成2年7月10日採録)

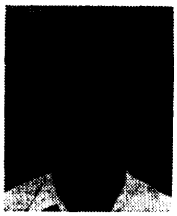
関口 智嗣（正会員）

1959年生．1982年東京大学理学部情報科学科卒業．1984年筑波大学大学院修士課程理工学研究科修了．同年電子技術総合研究所入所．計算機アーキテクチャと数値解析の研究に従事．特に科学技術計算用並列アルゴリズムに興味を持つ．市村賞受賞．現在，データ駆動計算機とスーパコンピュータ評価技術の研究を行っている．日本応用数理学会，日本ソフトウェア科学会各会員．



島田 俊夫（正会員）

昭和20年生．昭和43年東京大学工学部計数工学科卒業．昭和45年東京大学大学院修士課程修了．同年電気試験所(現電子技術総合研究所)入所．現在，同所計算機方式研究室室長．コンピュータグラフィックス，人工知能向き言語，LISP マシン，データフロー計算機の研究に従事．高度な並列処理方式に興味がある．電子情報通信学会，IEEE 各会員．



平木 敬（正会員）

昭和51年東京大学理学部物理学科卒業．昭和57年同大学院理学系研究科博士課程修了．同年電子技術総合研究所入所．理学博士．計算機アーキテクチャ全般，特にリスト処理計算機，データフローマシン，スケジューリングなどの研究に従事．元岡賞，市村賞各受賞．情報アーキテクチャ部計算機方式研究室主任研究官．現在 IBM ワトソン研究センター招聘研究員．