解　説

データ駆動計算機用の高級言語と処理系†

関口智嗣††　山口喜教††

1. はじめに

並列処理による高速計算を目標として，さまざまな計算機アーキテクチャが提案され，また試作されてきた．このような中でデータ駆動計算機アーキテクチャはデータ駆動計算モデルが命令，文，関数などの多様な粒度に応じて並列性の抽出を統一的に扱えるため，将来の並列処理において不可欠なアーキテクチャ技術である．

データ駆動計算機の研究はこれまでにアーキテクチャの側面から各種のデータ駆動計算機が試作され，また，ソフトウェアの側面からデータ駆動計算機に適した言語の研究がなされてきた．この両者に共通する基盤のアイデアがデータ駆動計算モデルとして位置付けられ，そのモデルを媒介にした言語からアーキテクチャへの写像は容易であったと言える．しかしながら，効率化，高速化を求められるデータ駆動計算機のハードウェアは必ずしもデータ駆動計算モデルの枠組に捉えられずに発展してきた．副作用を許さない純粋なデータ駆動計算モデルに基づく言語ではこのような計算機ハードウェアの性能を十分に引き出すことが不可能である．

本稿においてデータ駆動計算機用高級言語とは関数型言語に単一代入規則を導入した種類の言語を基本とするが，これを拡張して手続き型言語からのコンパイル技術について述べる．さらに，ユーザにとってデータ駆動計算機における高級言語を従来のデータ駆動計算機や並列フォンノイマン計算機における記述に対してより強力なものとする立場から解説を行う．以下，データ駆動計算機の実行モデルを述べ，データ駆動計算機言語の特徴，実際の言語，処理系における特徴を順に述べる．

† High Level Language for Data Flow Computer and its Compilation by Satoshi SEKIGUCHI and Yoshinori YAMAGUCHI (Electrotechnical Laboratory).

†† 電子技術総合研究所

2. データ駆動計算モデルと言語

2.1 データ駆動計算モデルとデータ駆動計算機

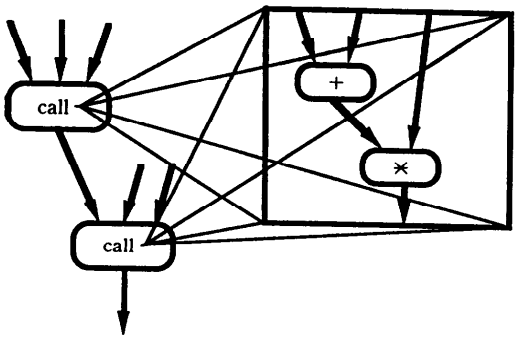
データの実行に関しては，ある演算に必要なデータが全て確定していることが唯一の条件である．この条件により，実行はデータ依存グラフをトレースしながら進むことになる．したがって，データ駆動計算機の実行はデータ依存グラフの記述によって定まる．

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方のトークン（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．以下では「グラフ」と記してデータ依存グラフのことを示す．

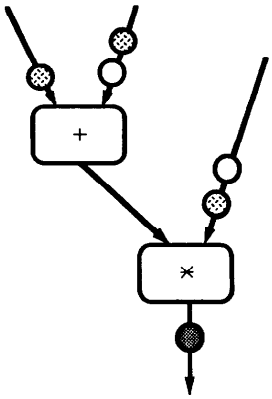
1974年に MIT の Dennis らのグループによって整理された[1] データ駆動計算モデルに基づいて実行を行う計算機をデータ駆動計算機と呼ぶ．データ駆動計算機を解説した成書として参考文献 2)〜4)をあげておく．

データ駆動計算機におけるプログラムの実行はグラフ上にトークンを流すことで行われる．関数呼び出しを例にとると，関数が呼ばれるたびに関数を記述したグラフのコピーを物理的に行う静的なモデルとトークンに色をつけることで区別しグラフの論理的なコピーを行う動的なモデルがある（図-1）．後者を特にタグ付きトークンデータ駆動計算モデルと呼ぶことがある．このモデルでは同一の色のトークンが入力アークに揃ったときノードが発火する．

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，



静的モデル



動的モデル

図-1 静的モデルと動的モデル

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．これを避けるため Dennis らは同一アーク上にトークンを一つしか置かず，実際にはコピーを行わないという制約を設けた．また，動的モデルではトークンに色をつけるため，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用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．

一般的なデータ駆動計算機の定義は待ち合わせの機構をハードウェアで実現した場合であり，計算モデルをソフトウェアで実現したものはこの範疇に含まない．待ち合わせを個々の命令単位で行うものを命令レベルデータ駆動計算機と呼び，関数のようにより大きな実行単位で待ち合わせを行うマクロデータ駆動計算機と区別する．本稿では単にデータ駆動計算機とした場合に命令レベルデータ駆動計算機を意味する．

データ駆動計算機のアーキテクチャに関しては，各国で研究開発が行われ英国マンチェスタ大学のデータ駆動計算機をはじめとして，いくつかの試作機が製作された．これらの詳細に関しては文献 5),6)を参照されたい．この中で電子技術総合研究所の SIGMA-1[7] はハードウェアレベルで純粋なデータ駆動制御をサポートしており，128 台の演算装置により 427 MFLOPS の最大性能を達成している．また，最近開発されているデータ駆動計算機には，データ駆動制御の利点を生かしつつデータ駆動制御をサポートするために生じる種々のオーバヘッドを克服するために新たな考え方が導入されている[8),9)]．

2.2 データ依存グラフ

グラフの基本構成要素には，図-2 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う．併合 (merge) ノードは複数の入力アークのうちいずれか一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) ノードは出力アークをもたない唯一のノードである．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-2 の場合，それぞれのノードは二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 本まで出している．これは命令セットアーキテクチャ上の制約による．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場合にはグラフの基本構成要素を組み合わせて，論理的には多入力多出力の複合ノードを考えればよい．

2.3 非巡回グラフと単一代入規則

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グラフにならない．ここでいう安全性の必要

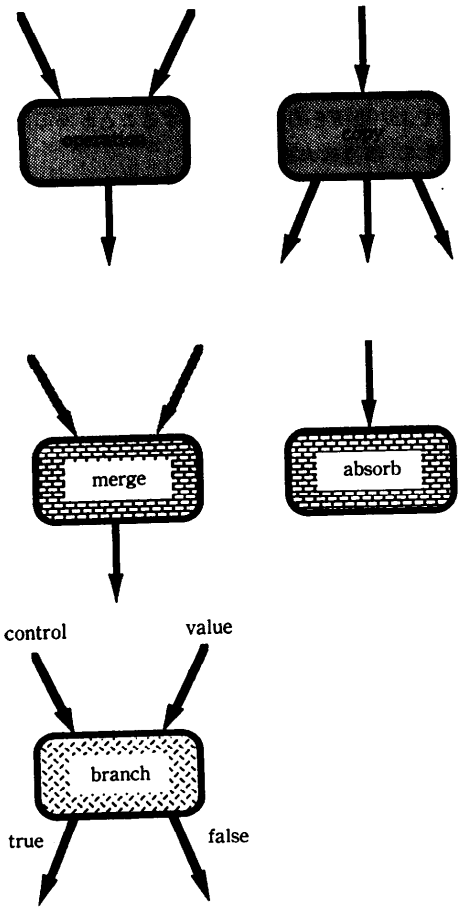


図-2 データ依存グラフの基本構成要素

十分条件は

1. 入力アークのないノードや出力アークのないノードが存在しない,

2. どのような入力値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，グラフ内部にトークンが残留しない,

ことである．1の条件は吸収ノードがないことと同値である．吸収ノードは発火してもその事実が残らないため発火を検出することができない．2の条件は，グラフの関数的な性質として入力値によらず，全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，グラフ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グラフ内部の必要なノードは全て発火したことを保証し，当該グラフを消去することが可能となる．

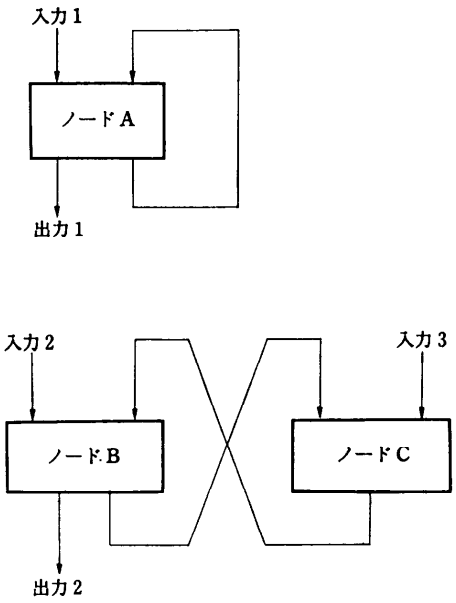


図-3 デッドロックが生じるグラフ

安全なブロックとなるための十分条件として，データ依存グラフの基本構成要素のうち，演算ノードと複写ノードのみで作成されるグラフを考える．

図-3 の例においては，ノードAの入力にはノードAの出力が必要である．このような依存関係がある場合，入力1を与えただけでは出力1は生成されない．また，ノードBとノードCに関して，入力2および入力3を与えても出力2は得られないというデッドロックが生じる．併合ノード，分岐ノードを含まないグラフにおいて巡回経路が存在することは自己依存するデータ依存関係が生じるため安全なグラフとはならない．安全なグラフとなるためには内部のデータ依存グラフに巡回経路が存在しないことが必要十分条件となる．

巡回経路が存在しないグラフにおいて，アークやノードにそれぞれ識別のための一意な名前をつけることが可能となる．たとえば，演算ノードにおいてノード名を f とし，入力アーク名を x，出力アーク名を y とする．このことを簡単に $y=f(x)$ と記述し，変数 y への代入式とする．また，コピーノードは入力のアーク名と出力アーク名を同一視する．

アークの名前が一意であることにより全ての変数に対する代入式は一度に限られている．このように変数

への代入が変数スコープの中で一度に限られる規則を単一代入則と呼ぶ．演算とコピーノードからなる安全なグラフならば単一代入則を満足する記述が可能となる．

しかしながら，字面上で単一代入則を満足するだけでは安全なグラフの記述が可能とならない．すなわち，

$$y=f(x) \quad (1)$$
$$x=g(y) \quad (2)$$

という記述は単一代入規則を満足しているが，図-3のデッドロックの例となる．したがって，通常の言語では変数の参照より前に変数の値定義が存在するという解釈順序を定義する．これにより，代入(1)ではxが値未定義変数となり，デッドロックが検出できる．

2.4 構造文の記述

非巡回グラフは安全なグラフであり，単一代入規則を用いて記述できる．しかし，演算ノードと複写ノードしか用いないため，実際のプログラムにおいては条件分岐やループ構造などを表現することができない．ここでは，if文で代表される条件分岐文とfor文で代表されるループ構造の単一代入規則による表現を考える．

単一代入則を適用した言語で変数aにすでに値が定義されているとする．さらに，if文において then 節では変数aに3という値を再代入したい場合を考える．このとき，単一代入規則であっても

$a=5$
if (P) then $a=3$

とaへの再代入を許す記述が考えられる．この場合はaという変数名で参照を記述できる．また，

$a=5$
if (P) then $a_0=3$ else $a_1=a$
$a_2=merge(a_0, a_1)$

として併合ノードを陽にして a_0, a_1 のいずれか一つが必ず定まることを示す記述がある．この場合はif文以降での参照でaとa_2は明確に区別される．さらに，a_0, a_1を参照することはできない．いずれにしてもaという変数のトークンは else 節にそのまま流し，then 節で新たに a_0 という変数に与えられた値とマージが行われる．

For 文の場合，繰り返し構造は分岐ノードと併合ノードを用いて記述することができる．$i=i+1$ のような記述の必要性はループ変数の更新など頻繁に生じる．これを単一代入規則を用いて記述することは非常に困難をともなう．そのため，実際の言語では new (next) や old といったキーワードにより現在のループ世代か一世代前かを明示したり，単一代入規則を破ってよい場所を例外的に設定することがある．

2.5 構造体の記述

データ駆動計算機における基本的なデータ構造としてストリームと配列がある．ストリームとは一次元的に並んだ値の列として定義されるデータ構造である．その値の生成と消費は逐次的に規定されているが，全ての値が同時に存在しなくてもよいという性質をもっている．ストリームの操作は関数性を保つためデータ駆動計算モデルとの親和性がよい．しかし，ストリームがデータトークンとして全て生成され，また順序を保証する必要があり，データ駆動計算機において効率のよい実現が困難である．

関数型言語に自然な形で配列を導入したのがArvindらのI構造である[10),11)]．これは書き込みが一度，読み出しが一度という配列で単一代入規則の拡張になっている．同一要素への再書き込みが発生すると古い配列を残したまま新たに配列全体をコピーし，当該要素を書き換える．このため，実現には古い配列の回収やコピーのオーバヘッドなどの問題点を解決する必要がある．

これに対して，I構造に自然な制限を加えた制限的I構造が提案された[12)]．制限的I構造におけるそれぞれの操作は

1. 書き込み操作によりデータの存在ビット(Pビット)を立てる，
2. Pビットが立っている要素に書き込み操作を行った場合はエラーとする，
3. 読み出し操作によりPビットが立っている場合は，そのデータを読み出してPビットを降ろす，
4. 読み出し操作の時点でPビットが立っていない場合，読み出し操作は保留される，

という関係を満たす．これにより，配列への読み書きの回数が実行時に決定するような操作が並列実行可能となった．

3. データフロー計算機用高級言語

ここでは，これまで考案，開発されたデータ駆動計算機用高級言語について具体的な例をあげる．

3.1 VAL

データフロー計算機の提唱者であるMITのDennis教授が中心になって開発したデータフロー計

算機用の高級言語で，Value Oriented Algorithmic Language の頭文字から名前が付けられている[13]．単一代入規則に基づいた関数型の言語であり，そのシンタックスは Pascal 風な汎用言語に近い．言語の特徴としては，逐次的な繰り返し文のほかに並列実行文を備えている点があげられる．条件文は Pascal 的な if then else の構文であり，繰り返し的な for 文を記述することも可能である．VAL による繰り返し文の例を示す．

```
for I: integer:=1; SUM: integer:=0;
do if I<=N then
                iter
                    SUM:=SUM+X[I];
                    I:=I+1;
                enditer;
   else SUM
  endif
endfor
```

この例では iter という構文の中では単一代入規則が破られている．

3.2 Id

Id は，当初カリフォルニア大学アーバイン校で，Arvind などによって提唱され[14]，その後 MIT で開発および改良が行われている[15]．この言語の特徴は，単一代入規則に基づいた言語でありながら，繰り返し構造を全面に押し出し，unfolding という機構によってこの繰り返し構造をデータ駆動計算モデル上で効率的にサポートしようとした点である．また構造体は I 構造である．Id による記述例を示す．

```
{sum=0
In
  {For i From 1 To n Do
    Next sum=sum+x[i]
  Finally sum}}
```

この例では繰り返し文を用いており，各ループ世代が重畳的に並列に実行可能であるとする．したがって各ループにおいて sum という変数はすべて異なって解釈されなければならない．このようにして単一代入規則をそのままの形で適応できないために，Id では Next（古いシンタックスでは new）という語で変数を修飾することによって，疑似的に単一代入則を満たすようにしている．

3.3 SISAL

Streams and Iteration in a Single Assignment Language の頭文字をとった言語である．この言語は，特にデータフロー計算機用として設計された言語ではなく，汎用的な並列処理言語としてコロラド州立大学，DEC，ローレンスリバモア国立研究所，マンチェスタ大学が共同で開発した言語である[16]．この言語は VAL を下敷きにしておりそのシンタックスは Pascal に近いが，繰り返し構造のサポートやストリームの扱いなど言語上のセマンティックスはむしろ Id に近い．SISAL で記述した例を次に示す．

```
for initial SUM:=0;
    I:=1
while I=<N
repeat
    SUM:=old SUM+X[I];
    I:=old I+1
return SUM;
```

この例においても，繰り返し文において，変数は各ループ世代において全て異なって解釈する．このために，SISAL では old という語で変数を修飾することによって，疑似的に単一代入則を満たすようにしている．

3.4 DFC と DFCII

DFC[17] は電子技術総合研究所において SIGMA-1 用に開発された高級言語で，大部分のシンタックスは C言語に準拠している．構造文としては if 文，for 文がある．変数については単一代入則を導入するが，for 文の式3に相当する増分節に限り単一代入則を破ることができる．プログラム例を示す．

```
for(i=0, sum=0; i<N; i=i+1, sum=sum+y){
    y=x[i];
}
```

DFCII[12] は SIGMA-1 用の高級言語として新たに設計された言語である．DFC と異なり，C言語の手続き的な記述に注目し，if 文，for 文，switch-case 文を実装した．goto 文と構造文の break 文，continue 文は解釈の原理的な可能性があるものの，goto の飛び先すべての組み合わせた場合を尽くす必要があるため処理系の実行時間が指数オーダで要することになる．さらに，動作はデータ依存関係による並列実行が基本とされている．また，ループによる色の枯渇を避けるため，逐次的なループ実行を実現するための syncfor 文やラベルを利用した sync 文による同期の記述が可能となっている．関数から複数の戻り値を記述できる点や複素数型のデータ構造がある点などでC言語からの拡張が施されている．

データ駆動計算機用高級言語としては単一代入則の撤廃を行った稀な言語である．通常のC言語と同様な記述が可能で，プログラム例を示すがC言語との違いが生じない．

```
for(i=0, sum=0; i<N; ++i)
  sum=sum+x[i];
```

3.5 記号処理用データ駆動計算機言語

記号処理を指向したデータ駆動計算機用の高級言語として Valid[18]，EMLISP[19] などがある．Valid は NTT の武蔵野電気通信研究所で開発された動的モデルに基づくデータ駆動計算機 DFM[20] をターゲットマシンとして開発された高級言語であり，タイプ付きの関数型言語を基本としているが，ユーザが並列評価，逐次評価などの制御を指定することができるためのプリミティブが用意されている．シンタックスは Algol に近い．プログラムの実行に際しては先行 cons によって，並列性を先行的に抽出する方式が採用されているが，特別なオペレータによって遅延評価を指定することも考慮されている．再帰式によりループ構造を記述する．このとき単一代入規則を破る変数は recur 式の中で指定する．Valid の処理系に関して詳細は文献 21)にある．

EMLISP は電子技術総合研究所で開発された記号処理向きのデータ駆動計算機 EM-3 をターゲットとして開発された Lisp 風の言語で，純 Lisp にブロック構造を導入することによって単一代入規則に基づいた言語を構成している．プログラムの解釈は，先行的な評価が想定されている．また，関数的な性質をもつ汎用的な言語，KRC[22]，FP[23] などもデータ駆動計算機用の高級言語として用いることも可能である．

4. データ駆動計算機における中間言語

高級言語の処理系を設計する場合，中間言語を設定する場合と直接オブジェクトコードを生成する場合がある．データ駆動計算機における中間言語の必要性を議論する前に，フォンノイマン計算機を対象とした処理系において中間言語を設定するメリットとデメリットを考える．メリットとしては，

1. コンパイラ作成の容易さ
2. 移植性の向上
3. 保守性の向上
4. 可読性の向上
5. 拡張性の向上

がある．デメリットとしては，

1. コンパイル速度の低下
2. 大域的な最適化が困難

といった点がある．

フォンノイマン計算機用の中間言語は P-CODE に代表されるような，仮想スタックマシンを対象とし，その中間言語を通常はレジスタマシンで模擬実行するという形式を取っている．したがって，実際のターゲットマシンでそれぞれ，中間言語の解釈を行うため，移植性，保守性は保たれるが，生成されたオブジェクトコードは，効率の悪いものとなる．ただし，仮想スタックマシンを対象とするため，コンパイラから中間コードの生成は非常に容易なものとなる．

最近は GCC[24] のように，中間言語でレジスタマシンを直接対象としたものもある．これは，マシンの特性を記述したファイルをコンパイラに渡すことにより，上記のメリットを保ったまま，デメリットを克服するような狙いがある．

データ駆動計算機用の中間言語は，SISAL においては IF 1[25]，Id における program graph[26]，DFCII における SASGA[27]，Valid における SIML[21] がある．IF 1 からは Manchester Dataflow Computer[28] や Cray-X/MP，Cyber-205 などへの移植が行われた．

フォンノイマン計算機用の中間言語は主に仮想スタックマシンを対象としていた．しかし，明らかに仮想スタックマシンの中間言語をデータ駆動計算機は効率良く模擬できない．IF 1 においては仮想スタックマシンを仮定せず，高級言語レベルの文に対応してデータ依存グラフを作成するものである．高級言語が変数に単一代入則を採用しているため，データ依存グラフ作成のためのデータ依存解析はコンパイラにとって非常に容易なものとなっている．ところが分岐構造やループ構造に対しては compound ノードとして Select ノード，LoopA ノード，LoopB ノードなどを作成しただけで，実際のデータ駆動計算機の実行に必要な，世代を跨る変数や，ループで不変な変数を検出することが困難になっている．

IF 1 はデータ駆動計算機に必ずしも特化していたわけではないので，計算機による命令セットの違いを吸収することができていない．したがって，データ駆動計算機個々の命令セットに依存せず，ループや分岐構造を容易に扱えるレベルでの中間言語が必要となる．そこで，データ駆動計算機用の中間言語には，中間言語が対象としている仮想マシンの定義を行う．仮

想マシンにおいては演算，複写ノードで記述できるデータ依存関係と分岐，併合ノードで記述できる制御の流れを明確に分離して記述する必要がある．

5. データ駆動計算機用言語処理系の特徴

実際のデータ駆動計算機用言語処理系の特徴をDFCII を題材にして概観する．DFCII の特殊性は極力避け，単一代入規則の言語に対しても適用可能な処理方法に限定した．なお，DFCII の特殊性を含めた処理系の詳細は文献 29), 30) に譲る．

5.1 基本構造文の処理

基本構造文とは式の並んだ文である．例として

$$x=x+y \qquad (3)$$

$$z=x\times 4 \qquad (4)$$

を考える．DFCII では x に関するデータ依存解析を行い，(4)で参照している x は(3)で値定義された x という解釈を与える．したがって，このグラフは図-4 に示すようになる．このグラフは安全であるから単一代入規則を用いても記述できる．なお代入を表す＝ノードは仮想的なノードで代入にともなう型変換が必要な場合に効果がある．

5.2 分岐構造文の処理

DFCII における 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-5 のように条件節ブロック，then 節ブロック，else 節ブロック，

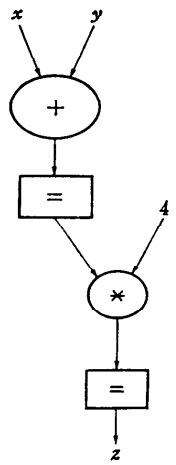


図-4 基本構造文の処理

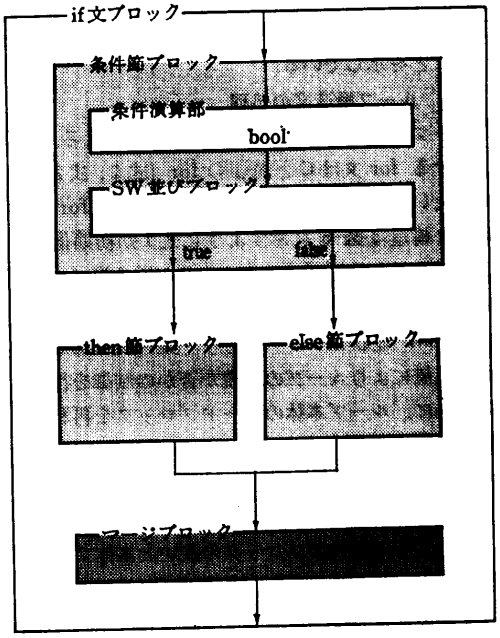


図-5 If 文の処理の流れ

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 並びブロックでは分岐ノードを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り，基本構造ブロックとしてグラフは生成される．マージブロックは if 文ブロック全体が安全なグラフ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する．

DFCII では単一代入則を適用しないため，全ての変数について

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，

を調べる．これらの性質の組合せにより，条件値が真の場合に参照される変数，偽の場合に参照される変数，常に参照される変数を決定することが可能であ

る．これは，変数の定義参照関係を調べることにより，単一代入規則を用いた記述と同様なグラフが得られることを示している．

5.3 ループ構造文の処理

ループ構造文として，for 文の処理を考える．DFCII における for 文は C と同じく 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-6 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，ループ本体のコードブロックを再利用しながら実行するためにトークンに色をつける I_INC ブロック，色をリセットするための I_CLR ブロックがある．

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，変数に関しての性質として，

1. for 文以前に変数への値定義がある，
2. 条件節での変数参照がある，
3. 条件節での変数の値定義がある，

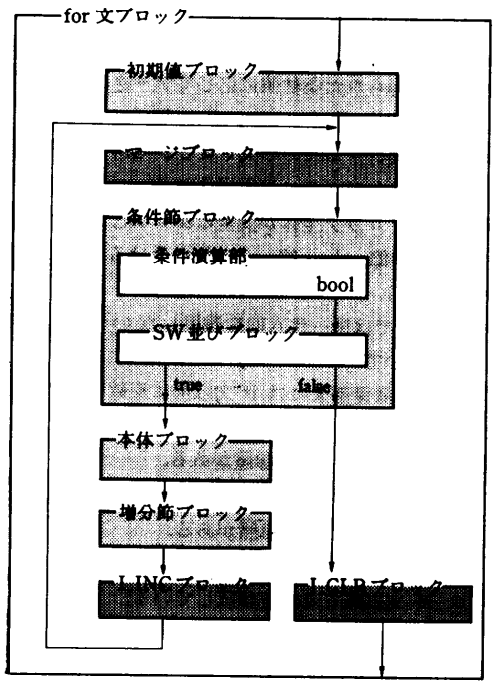


図-6 For 文の処理の流れ

4. 本体節での変数参照がある，
5. 本体節での変数の値定義がある，
6. for 文以降で変数を参照している，

の組合せにより，変数の値が参照される場合が定まる．

I_INC ブロックに現れる変数が単一代入規則を破って記述されている変数と同じであることを注意してほしい．この事実により new や old といった修飾子がなくても DFCII においては変数の定義参照関係を調べることで安全なグラフが作成できる．

なお DFCII は while 文ほかのループ構造文があるが，for 文の特殊な例であるので，容易に類推できる．

5.4 同期トークンの処理

最適化されていないプログラムでは値定義がなされたものの参照されなかったトークン，配列へ書き込みにいった際に生じる確認のトークン，分岐ノードを実行したときに生じる利用されないトークンが存在する場合がある．あるコードブロックの終了を検知するためにはグラフ内にトークンが留まらないようにする必要がある．このため，値が用いられなかったトークンを全て回収する．

このようにして，1個になったトークンを同期トークンと呼び，発生元によってさまざまな情報を担っている．特定のコードブロックの終了信号とも考えられ，データ駆動計算機においては唯一の実行の流れを表現することのできる重要なトークンである．

同期トークンの処理には DFCII では sync ノード，Id では gate ノードを用いる．Sync ノードが発火すると第一入力のトークンを出力アークに流すノードである．このノードにより参照されなかったトークンの数が1ずつ減少する．

この同期トークンの回収処理はループ構造文において複雑なものとなる．すなわち，ループ文においては各ループ世代の並列実行が可能であれば，並列実行を妨げないコードを生成する必要がある．Id では各ループ世代ごとで発生した同期トークンと論理値に gate ノードを設け同期させるため[26]同期トークンの回収が終了するまで次のループ世代に進むことができず並列実行を妨げる．並列実行を行うループにおいては同期トークンは各ループ世代から独立して発生させることにより本体部の繰り返しとは無関係にこれらを集めて最後に一つのループ終了を示す同期トークンとする必要がある．SIGMA-1 における実現方法は文献 30) にある．

5.5 最適化

データ駆動計算機用の最適化は命令実行時間を基準に行われる．これには，命令数を減少させることにより並列度を犠牲にする最適化もあれば，命令数を増加させても並列処理により命令実行時間の短縮が実現できる場合のようなトレードオフがある．具体的には次のような点が考えられている．

1. 命令の複合化による実行命令数の減少
2. 関数呼び出しのループへの変換
3. データ構造操作の配列による最適化
4. 分岐処理命令の最適化
5. ノードの割り付け
6. 非参照トークンの削除
7. ループ不変変数の先行評価
8. 高機能構造体によるベクトル処理

1はループにおける添字判定部分，構造体の読み出し部分，ループにおける同期トークンの処理などにおいて定型的な処理形態となる命令を複合化するものである．このような最適化により実行命令数を減少させることが可能である．ただし，複合化された命令を適用できるかどうかの判定は処理系の負担となる．

2はテールリカージョンで記述できる関数呼び出しの場合にループ構造へと変換するものである．これにより，関数呼び出しにおけるトークンの色の獲得や引数の受渡しにおいて生じるオーバヘッドを削除することができる．

3はストリーム操作の一部を配列として実現することにより演算の施されないストリーム要素がトークンとして流れることを抑制することが可能とするものである．

4は分岐構造文やループ構造文の処理における最適化である．通常，分岐ノードに入力されたトークンは制御入力の真偽値にしたがって，必ず出力ノードの一方へ選択的に出力される．しかし，実際のプログラムではそのトークンを必要としない場合がある．そのトークンが利用されない場合は非参照トークンとして回収の対象となる．しかし，入力アークの制御トークンを常に出力し，データトークンは必要に応じて出力するという分岐ノードを考える[29]と，分岐ノードから発生した制御トークンは他の分岐ノードの入力とすることが可能となるため非参照トークンを減少できる．ただし，従来の分岐ノードは全て並列に実行可能であるが，ここでは，制御トークンを順送りするため一部で逐次化される．非参照トークンの回収処理と並列化による速度向上とのトレードオフになる．

5はノードの演算装置への割り付けによるデータ転送オーバヘッドの削減である．実行前にプログラムなどの性質により静的な割り付けを行う．割り付けの準最適なアルゴリズムは文献 31)などがある．

6はループ内で参照しか行われない変数に対してループごとに変数の値を評価しないようにするものであり，フォンノイマン計算機におけるループの最適化としても知られている．変数の評価式が副作用を含まないような演算で構成されていれば，その評価をループ文の実行以前に行い，値を定数として与える．これはグラフから検出してコンパイラがグラフの変更を行うべきである．

7はデータ駆動計算機におけるベクトル演算の実装である．配列の指定した部分に関して演算を連続的に行うことができる．SIGMA-1 において高機能構造体として実現した[32]．これにより，添字の生成やループの色の更新命令などが削除された．

6. おわりに

並列計算機として今後の研究が注目されているデータ駆動計算機における言語とその処理系について概観を述べた．データ駆動計算機特有の命令を用いて分岐やループを表現する手法を述べることによってデータ駆動計算機への処理系の特徴とした．データ駆動計算機用言語として関数型言語をもとにした研究と手続き型言語をもとにした研究がある．二者択一である必然性はなく，近い将来データ駆動計算機も普通の計算機と同様に考えられるようになるであろう．

謝辞 本稿執筆にあたり，島田俊夫，坂井修一，西田健次の各氏より貴重なコメントをいただいた．ここに感謝する．

参考文献

1) Dennis, J. B.: First Version of a Data Flow Procedure Language, Lecture Notes in Computer Science, Springer Verlag, Vol. 19, pp. 362-376 (1974).
2) Sharp, J. A.: Data Flow Computing, Ellis Horwood Ltd. (1985).
3) 富田眞治(訳)：データフローコンピューティング，サイエンス社 (1987).
4) 曽和将容：データフローマシンと言語，昭晃堂，(1986).
5) Veen, A. H.: Dataflow Machine Architecture, ACM Comput. Surv., Vol. 18, No. 4 (1986).

6) 島田俊夫：数値計算向きデータフローマシン，情報処理，Vol. 26, No. 7, pp. 780-786 (1985).
7) Hiraki, K., Nishida, K., Sekiguchi, S., Shimada, T. and Yuba, T.: The SIGMA-1 Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems, J. Inf. Process., Vol. 10, No. 4, pp. 219-226 (1987).
8) Yamaguchi, Y., Sakai, S., Hiraki, K., Kodama Y. and Yuba, T.: An Architectural Design of a Highly Parallel Dataflow Machine, Proc. IFIP Congress '89, pp. 1155-1160 (1989).
9) Iannucci, R. A.: Toward a Dataflow/von Neumann Hybrid Architecture, Proc. 15th. Ann. Int. Conf. on Comput. Arch., pp. 131-140 (1988).
10) Arvind and Thomas, R. E.: I-structures: An Efficient Data Type for Functional Languages, TM 128, LCS, MIT (1979).
11) Arvind, Nikhil, R. S. and Pingali, K. K.: I-Structures: Data Structure for Parallel Computing, ACM Trans. on Prog. Lang. and Systems, Vol. 11, No. 4, pp. 598-632 (1989).
12) 関口智嗣，島田俊夫，平木 敬：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).
13) Ackerman, W. B. and Dennis, J. B.: VAL-A Value Oriented Algorithmic Language: Preliminary Reference Manual, TR 218, LCS, MIT (1979).
14) Arvind, Gostelow, K. P. and Ploufe, W.: An Asynchronous Programming Language and Computing Machine, TR 114 a, Dept. Comp. Sci., Univ. Calif., Irvine (1978).
15) Nickhil, R. S.: ID (Version 88. 0) Reference Manual, TR 284, LCS, MIT (1988).
16) McGraw, J., et al.: SISAL: Streams and Iteration in a Single Assignment Language, TR-M-146, Lawrence Livermore National Lab. (1985).
17) 島田俊夫，関口智嗣，平木 敬：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. J 71-D, pp. 501-508 (1987).
18) 長谷川隆三，雨宮真人：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. J 71-D, pp. 1532-1539 (1988).
19) 山口喜教，戸田賢二，弓場敏嗣：先行制御機構を持つデータ駆動計算機 EM-3 の評価，電子情報通信学会論文誌，Vol. J 72-D-I, No. 3, pp. 182-195 (1989).
20) Amamiya, M., Takesue, M., Hasegawa, R. and Mikami, H.: Implementation and Evaluation of a List-Processing Oriented Data Flow Machine Architecture, Proc. 13th, Ann. Int. Symp. on Comput. Arch., pp. 10-13 (1986).
21) 雨宮真人，長谷川隆三：データフローマシン DFM と関数型プログラミング言語 Valid, bit 臨時増刊, Vol. 21, No. 4, 共立出版, pp. 511-520 (1989).
22) Turner, D. A.: The Semantic Elegance of Applicative Languages, Proc. Symp. on Funct. Prog. Lang. and Comput. Arch., pp. 85-92 (1981).
23) Backus, J.: Can Programming be Liberated from the von Neumann style? A Functional Style and its Algebra of Programs, Comm. ACM, Vol. 21, pp. 613-641 (1978).
24) Stallman, R. M.: Internals of GNU CC.
25) Skedzielewski, S. K. and Glauert, J. R. W.: IF 1 An Intermediate Form for Applicative Languages, TR5M-170, Lawrence Livermore National Lab. (1985).
26) Traub, K. R.: A Compiler for the MIT Tagged-Token Dataflow Architecture, Master Thesis, MIT (1986).
27) 関口智嗣，島田俊夫，平木 敬：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).
28) Gurd, J., Kirkham, C. C. and Watson, I.: The Manchester Prototype Dataflow Computer, Comm. ACM, Vol. 21, No. 1, pp. 34-52 (1985).
29) 関口智嗣，島田俊夫，平木 敬：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，電子情報通信学会技術研究報告，SS 89-26, pp. 11-18 (1990).
30) 関口智嗣，島田俊夫，平木 敬：データ駆動計算機 SIGMA-1 用並列記述言語 DFCII の処理系と評価，並列処理シンポジウム JSPP '90 講演予稿集 (1990).
31) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, IEEE Trans. Comput., Vol. C-33, No. 11, pp. 1023-1029 (1984).
32) Hiraki, K., Sekiguchi, S. and Shimada, T.: Efficient Vector Processing on a Dataflow Supercomputer, Proc. SUPERCOMPUTING '88, pp. 374-381 (1987).

(平成2年3月28日受付)