ループ間にまたがるデータ参照関係をもつ多重ループの自動ベクトル化†

津田孝夫†† 国枝義敏††
二宮正和†† 粟屋透††*

現在日本のメーカ製作によるベクトル計算機の自動ベクトル化コンパイラは，任意多重ループの異なるループにまたがるデータ参照関係を検査できず，ベクトル化率向上の障害となっている．本論文は，この困難を解消する新しい方法について述べたものである．基本変数を定義し，これを用いてすべての配列変数をベクトル間接参照によりアクセスすることにより，任意の多重ループを等価な1重ループに変換する．これを限界までベクトル化する．必要なアルゴリズムを導入し，またプリプロセッサとして実現した結果の実測例も紹介している．

1. はじめに

際限のない大規模科学技術計算の需要に応えるため，わが国の計算機メーカから並列パイプライン方式にもとづくベクトル計算機が提供されるようになった．S 810/20（日立製作所製），VP-100（富士通製）は，現在それぞれ東京大学，京都大学大型計算機センターに設置され，一般ユーザはそれらを使うことができる．1985年中には SX-2（NEC 製）が市場に登場するといわれている．これらの各機種のおもな特長と性能については，海外でも調査報告がなされている[1]．これらのベクトル計算機はそれぞれ自動ベクトル化コンパイラ[2]によりサポートされている．ユーザは自分のプログラムを標準的な FORTRAN 77 で書くと，コンパイラはできるだけベクトル化可能な部分をみいだし，ベクトル機械命令に変換する．しかしベクトル化できない残りの部分は通常のスカラ機械命令に展開される．計算機は一般にベクトル命令とスカラ命令の混在した命令列を実行する．このようにプログラムの実効的性能は，コンパイラの性能に大きく左右され，十分高いベクトル化率を保証しない限り，ベクトル計算機のハードウェア性能を引き出せない．

現在メーカ提供の自動ベクトル化コンパイラの限界の一つは，多重ループにおいて深さの異なる複数個のループ，または同じ深さの異なる複数個のループにまたがるデータ参照関係を完全には検査できず，そのため一般の多重ループをベクトル化できないことである．現在ベクトル化できる多重ループは，宣言で添字範囲を定義した配列要素をその範囲でもれなく全点総なめして走査する tight な多重ループや，最内側ループでないループの文中に現れる変数としてはそのループにしか現れない（他のループにまたがらない）単純変数と配列変数に限られる，などきわめて限定されたタイプの多重ループに限られ，実用上障害となっている．

本論文においてはじめて，任意の多重ループを自動的にベクトル化する方法を提案する．ただし任意ではあるが GO TO 文を含まないものとする．IF-THEN (-ELSE) 文が含まれる場合にも拡張できる．また多重ループの各ループの制御変数の上・下限およびその増（減）値（インクリメント）は，そのより外側の（一般に複数の）ループの制御変数の任意の関数で，最外側ループの制御変数の上・下限とそのインクリメントはコンパイル時に値が確定しているものとする．

本論文で提案する方法の基本は，多重ループの制御変数を配列化し，配列要素を“基本変数”により間接参照して，多重ループを1重化してしまう点にある．複雑なデータ参照関係の多重ループは，現状ではほとんどベクトル化されないから，得られた1重ループのプログラムをベクトル実行すると，1重化のため生じたオーバヘッドを超えて原プログラムは高速実行されることが可能となる．原理的にはベクトル計算機の機種には依存しないが，ベクトル間接参照（リストベクトル）の性能が長いベクトル長に対し十分良くなくてはならない．内側のループの制御変数の上・下限が，

† Mechanical Vectorization of Multiply Nested DO Loops with Inter-Loop Data Dependences by TAKAO TSUDA, YOSHITOSHI KUNIEDA, MASAKAZU NINOMIYA* and TOHRU AWAYA** (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室
* 現在 マツダ(株)勤務
Now with Mazda Automobile Co.
** 現在 大阪市役所勤務
Now with the Osaka Municipal Office.

より外側の制御変数の値に依存しないような単純な場合には，1重化することにより最終の1重ループのベクトル長は，もとの多重ループの各ループ長の積になる（しかし一般には積にならない．後述図6の例参照）．そのような積の程度の長さのベクトル長に対し，間接参照が直接参照の2ないし3倍程度の時間ですむならば本論文の方法は有効である．本論文では方法の概略について述べる．実現上の各種技術については別論文を用意している．

2. 任意多重ループの1重化手法

2.1 S（スカラ）実行とV（ベクトル）実行におけるデータ参照関係

ある配列要素が代入文の左辺に現れるとき「定義」される，右辺に現れるとき「引用」されるという．この両者を含めてデータ参照関係という．ある配列変数の同一要素が2度出現するとき，(定義，引用)，(引用，定義)，(定義，定義)，(引用，引用) の順序で参照関係を生じうるが（これをデータ参照関係順序対あるいはたんに2出現という），この最後のものは値の変更をまったく伴わないから考慮する必要はない．またif文の条件節に現れるときは引用として扱う．3回以上出現するときは，この順序対のすべての可能な組合せで取り扱える．

スカラ命令群をベクトル命令群で置きかえて実行してもプログラムの意味が変わらないことは，すべてのデータ参照関係の順序対が保存される（すなわちスカラ実行でもベクトル実行でも不変である）ことと等価である．

2.2 多重ループの構造と基本（制御）変数

一つの多重ループを構成する各ループ間の親子関係（すなわち入れ子構造の関係）および兄弟関係（すなわち同一ループ内に同レベルで並置される二つ以上のループの関係）は，一つのループを一つの節に対応づけた木構造の節の親子関係および兄弟関係で表現できる．

ソースプログラムの構文解析の段階で，この構造は機械的に認識される．まったく兄弟をもたない（すなわちまったく分枝のない木で表される）多重ループを**構造のない多重ループ**，それ以外を**構造のある多重ループ**と呼ぶ．

もともと与えられている各文の実行順序を変えずに，実行とともに変化する各制御変数の値を配列化する．これも構文解析の段階で自動的に行える．前述のように，各制御変数の上・下限とインクリメントは，より外側ループの制御変数の任意の関数である．ただし最外側ループの制御変数とそのインクリメントは，コンパイル時に値が確定しているものとする．以上のように配列化した制御変数の添字を t とすると，構造のない多重ループの場合，t は1から増値1で，各ループを1回実行するたびに増す．この t により多重ループは1重化されるので，t を**基本制御変数**あるいはたんに**基本変数**と呼ぶ．構造のある多重ループに対しては，上述の木構造の最深部ループごとに基本変数を別々に定義する．

上述のように最外側ループの制御変数の上・下限とその増(減)値は，コンパイル時に値が確定しているという制限はあるが，その限りにおいて，基本変数によりすべての任意の多重ループは1重化される．

2.3 Diophantus 方程式の解による inter- および intra-loop データ参照関係の検査

インプリメンテーション上は構造のある多重ループの1重化プログラムを用意し，構造のない場合はその特別な場合として処理するようになっているが，説明をわかりやすくするため，まず構造のない多重ループについて述べる．

2.3.1 構造のない多重ループの1重化

Diophantus 方程式 構造のない多重ループに含まれる k 次元配列 A を考える．可能な2出現は

$$\left.\begin{array}{r}\cdots\cdots\\ A(f_1,\cdots,f_k)=\cdots\cdots\\ \cdots\cdots\\ =A(f'_1,\cdots,f'_k)\\ \cdots\cdots\end{array}\right\}\quad(a)$$

$$\left.\begin{array}{r}\cdots\cdots\\ \cdots\cdots=A(f_1,\cdots,f_k)\\ \cdots\cdots\\ A(f'_1,\cdots,f'_k)=\cdots\cdots\\ \cdots\cdots\end{array}\right\}\quad(b)$$

$$\left.\begin{array}{r}\cdots\cdots\\ A(f_1,\cdots,f_k)=\cdots\cdots\\ \cdots\cdots\\ A(f'_1,\cdots,f'_k)=\cdots\cdots\\ \cdots\cdots\end{array}\right\}\quad(c)$$

で，(a)(定義，引用)，(b)(引用，定義)，(c)(定義，定義) がある．ただし引用される配列要素に対しては一般に演算が施されるが，ここではたんに右辺にその配列要素が書いてある．またS実行で先行しない

ほうの（すなわち文番号* が大きいほうの文中の）添字に prime を付した．同一文中の両辺に，同一配列名が現れる場合については別に後述する．

さて $A(f_1, \cdots, f_k)$, $A(f'_1, \cdots, f'_k)$ がそれぞれネストの深さ p, q のループに現れたとすると，添字 f_i および f'_i $(i=1, 2, \cdots, k)$ はそれぞれ元来与えられている制御変数 $I_1, \cdots, I_p$ および $I_1, \cdots, I_q$（ただし最外側から内側に向けて番号づけするとする）のみの関数で，参照関係が存在することは

$$\left.\begin{aligned} f_1 &= f'_1, \\ f_2 &= f'_2, \\ &\cdots \\ f_k &= f'_k \end{aligned}\right\} \qquad (1)$$

が成り立つことを意味する．これは $(p+q)$ 元の Diophantus 方程式（不定方程式）で，解は $(I_1, \cdots, I_p; I'_1, \cdots, I'_q)$ の整数値の組で与えられる．一方，各制御変数はすでに述べたように，基本変数 t の関数であるから，この整数値の解を実現する

$$\left.\begin{aligned} &I_1 = I_1(t), \cdots, I_p = I_p(t); \\ &I'_1 = I_1(t'), \cdots, I'_q = I_q(t') \end{aligned}\right\} \qquad (2)$$

なる基本変数の対（つい）(t, t') の存在をその可能なすべてについて各制御変数の配列化データから検索すれば，inter-loop（すなわち異なるループにまたがる）および intra-loop（すなわち同一ループ内）データ参照関係を検査できる．(1)の Diophantus 方程式はすべての整数解 $(I_1, \cdots, I_p; I'_1, \cdots, I'_q)$ をもれなく，しかも誤差ゼロで数値的に求めなければならない．

上述の各制御変数の配列化データとは，各制御変数が基本変数 t の増加とともに，どのような整数値をとるかを示す表である．これらの整数値を保持するため，基本変数 t の値域を $1, 2, \cdots, T$ とすると，(2)に対応して $T \times \mathrm{Max}(p, q)$ 個分程度の整数値のための空間的オーバヘッドが生ずるのはやむをえない．また配列化データから，(1)のすべての解 $(I_1, \cdots, I_p; I'_1, \cdots, I'_q)$ に対する可能な (t, t') 値の検索は，一見時間的オーバヘッドが大きいように思われるが，そうではない．最外側ループの制御変数 I_1 の下限，上限，ステップサイズを n_l, n_u, stepsize1 とし，

$$\lceil (n_u - n_l + 1)/\mathrm{stepsize1} \rceil \equiv \lambda$$

とすると，ヨコ方向が平均 T/λ，タテ方向が Max (p, q) の整数値の表（これを行列とみれば，その第1行は stepsize1=1 のとき $I_1 = 1, 1, \cdots, 1, 2, 2, \cdots, 2, \cdots$ のようになり，その同じ整数が続いて並ぶのが平均 T/λ 個となる）のタテ方向の並びの一致の検索を(1)の解の各 $(I_1, \cdots, I_p)$ および $(I'_1, \cdots, I'_q)$，すなわち(1)の解の個数の2倍の回数だけ行えばよい．

埋没定義と C 行列　(1)および(2)から(イ)解 (t, t') がない，か(ロ)1組ないし1組以上の解 (t, t') が存在する，かである．(イ)の場合データ参照関係はない．(ロ)の場合を詳しく検討してみよう．解 (t, t') の t 値に対応する配列要素が出現する文番号を s, t' 値に対応する配列要素が含まれる文番号を s' とする（ただし $s \neq s'$．$s = s'$ の場合については別に述べる）．次のように場合分けをする．

［A］ 得られた解の組 (t, t') のすべてについて $(t > t') \cup ((t = t') \cap (s > s'))$ が真である．

［B］ 得られた解の組 (t, t') のすべてについて $(t < t') \cup ((t = t') \cap (s < s'))$ が真である．

［C］ 上記［A］，［B］以外．

［A］は，着目した同一名の配列の参照関係において，S 実行においてつねに文 s' の実行が先行していることを意味し，［B］は同様につねに文 s の実行が先行していることを示す．［C］の場合は，その配列のいくつかの要素に対し“**埋没定義**”が生ずる．この埋没定義とは，多重ループを1重化して，得られた1重ループをそのままベクトル化すると，(引用，定義)ないし(定義，引用)の場合，ベクトル命令実行中に同じ配列要素に対する引用と定義の時間的順序が S 実行の場合と異なってしまい，引用に必要な正しい値を破壊してしまうことである．ループ中の単純変数を配列化して，必要な値を残してベクトル化する技術[3]に倣って，この場合はとにかく定義文を先行させ，ただし定義前の必要な値は退避させる．埋没定義は(定義，定義)の場合も起こることがあり，そのときは正しく定義が行われるよう分岐処理を付加する．

埋没定義の有無の弁別は，後述するように **C 行列** (collision matrix，衝突行列) を用いて機械的に行う．これは人間の手ではできないし，その有無はソースプログラムから直観的に看取できるものではない．埋没定義の実行時処理は時間がかかる．これを工夫により高速化を図るが，幸いすなおに書かれた現実のプログラムでは埋没定義の起こるチャンスは非常に少ないことがわかっている．

以上の検査を最外側ループ内に現れるすべての配列のデータ参照関係順序対について行う．Diophantus 方程式(1)に解があると判定された(ロ)の場合につい

* 以下文番号とは，多重ループでその DO 文や CONTINUE 文などの制御文を除いた文に付した内部文番号を指す．ラベルとして用いる番号ではない．

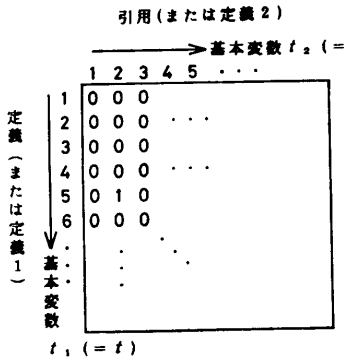

図 1 構造のない場合の C 行列
Fig. 1 C-matrix for the case with no structures.

解 {(t, t')} に関する場合分け				D 行列定義
(イ)	解 (t, t') が存在しない			——
(ロ)	[A]	解の組 (t, t') のすべてに対し (t>t') ⊔ ((t=t') ⊓ (s>s')) が真		$d_{s's}\leftarrow 1$
	[B]	解の組 (t, t') のすべてに対し (t<t') ⊔ ((t=t') ⊓ (s<s')) が真		$d_{ss'}\leftarrow 1$
	[C]	$s\neq s'$ で上記以外	(定義, 引用) 定義の文番号 d, (引用, 定義) 引用の文番号 q	$d_{dq}\leftarrow 1$
			(定義, 定義) Max(s, s') = max, Min(s, s') = min	$d_{s,\max}\leftarrow 1$

図 2 $s\neq s'$ の場合の解 $\{(t,t')\}$ の分類と D 行列の定義
Fig. 2 Classification of solutions $\{(t,t')\}$ $(s\neq s')$ and the definition of D-matrix.

て，次のような **C 行列**を作る．参照関係順序対が(定義，引用)，(引用，定義) のいずれの場合もタテ方向に定義を，ヨコ方向に引用をとり，基本変数は上から下に，また左から右に増加するものとし，行列要素はゼロまたは1とする．(定義，定義) の場合も同様で，その順に定義1および定義2としてそれぞれタテおよびヨコ方向に対応させる．たとえば**図1**に例示したように，配列 A の参照関係順序対のうち (引用，定義) について調べるとき，C 行列の第5行第2列に1が立つとは，基本変数が $t=5$ のとき定義が，$t=2$ のとき引用が同一の配列要素に対してなされることを示す．このように参照関係の衝突が同一の配列要素に対し生ずるとき1を立て，それ以外はゼロとする．C行列は概念上はきわめて大きいビット・マップとなるが，それをそのまま記憶するわけでなく，実際は(2)を満たす基本変数の対の集合 $\{(t, t')\}$ のみ保持する．またこの $\{(t, t')\}$ を求める時間的コストについては，2.3.1 項 Diophantus 方程式の項の末尾で述べたが，C行列の構造は多くの場合(準)周期性をもち，小行列部分のみを調べれば，他の部分は計算により1が立つ要素を推論できるので，さらにオーバヘッドをへらせる．C 行列は処理アルゴリズム構成上重要な役割を果たす．

C行列の対角要素全体を“境界”と呼ぶ．構造のある多重ループの場合，この境界は拡がりをもつ (後述参照)．境界より上にある要素全体を上三角，下にあるものを下三角と名づけると，下三角と上三角の両方に1が立つとき埋没定義が存在する．

D行列の作成 多重ループを1重ループに変換し，それをさらに可能な限界までベクトル化する必要がある．そのために **D行列** (data-referencing matrix, データ参照行列) を用いて文の移動などの操作をする．考えている多重ループのループ本体 (DO 文や CONTINUE の制御文は除く) の文の総数が l のとき，D 行列 (d_{ij}) は $l\times l$ 行列であって，たとえばその第2行第4列の要素 d_{24} が1であるとは，ソーステキストでの第2文が第4文よりS実行においてつねに先行するか，または埋没定義の処理などアルゴリズム (**図2**参照) で指定するようにつねに先行させるようにすることを示す．1の値をとらない要素はすべてゼロである．

C 行列の構成に応じて D 行列を作成するが，その作成は図 2 に従って行う．最終的には D 行列をさらに操作して文の移動などにより1重化されたループ自体のベクトル化率を限界まで高める．これについては後述する．図 2 による D 行列作成法を**アルゴリズム1 (異なる2文にまたがるデータ参照関係の検査)** と呼んでおこう．

以上は $s\neq s'$ の場合であるが，参照順序対 (2出現) が同じソース文で生ずる場合を次に述べる．このときは (定義，定義) はありえず，(引用，定義) のみである．このような $s=s'$ の場合については次の方法に従えばよい．

アルゴリズム2 (同一文中におけるデータ参照関係の検査) C 行列で1が立つのが

(a) 境界のみ，
(b) 下三角と境界上，
(c) 下三角のみ

のいずれの場合もそのまま V 実行可．

(d) 上記以外

のときは再帰的定義が生じていて V 実行不可． □

このような同一文中の 2 出現に対しては D 行列に登録する必要はない．その文は他と独立に判定され，後述の文のならべかえと無関係に判定に従って実行される．

2.3.2 構造のある多重ループの1重化

構造のある多重ループの場合，異なる深さのループ

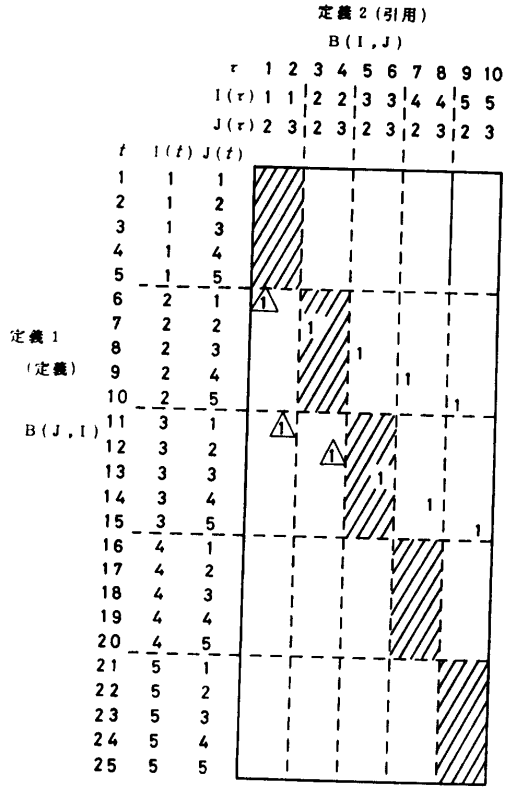

図 3 構造のある場合の C 行列
Fig. 3 C-matrix for the case with structures.

間，同じ深さの異なるループにまたがる同一配列の2出現（参照関係順序対）と同一ループ内2出現について検査する必要がある．

さて，同一名の配列の2出現につき，各出現の属するループがきまる．すなわち既述のループの木構造の節点がきまる．異なる二つのループすなわち節点の場合，浅いほうのループ（木の根に近いほうの節点）のレベルまで根の方向に深いほうのループから出発して木を辿ったとき，レベルが浅いほうのループに到達できれば構造のない場合の参照関係の検査法が適用できる．同一ループ内の2出現についても同じ検査法でよい．もし到達できなければ，両方の節点からさらに根に向かって辿り，最初の共通の分岐節点に達する．すでに述べたように構造のある場合には，最深部（木の葉）のループごとに別々に基本変数を定義している．いまそれらを t と τ，ただいまの分岐節点に対応するループの制御変数を I とすると，直接 t, τ で比較する必要はなく，$I(t)$ と $I(\tau)$ で先行実行かどうかの時間的順序を比較すればよい．たとえばかんたんな例として

```
      DO 30 I=1,5
        DO 10 J=1,5
          B(J,I)=0.0……定義 1
10      CONTINUE
        DO 20 J=2,3
          B(I,J)=1.0……定義 2
20      CONTINUE
30    CONTINUE
```

を考えると，C 行列は**図 3** のようになる．すなわち基本変数 t, τ によらず共通の親のループの制御変数 $I(t), I(\tau)$ で比較すればよく，それらが互いに同じ値をとる図3の斜線部が“境界”となる．このような境界の拡大解釈により，C 行列の上三角と下三角の両者に1が立つと埋没定義が存在することがいえる．図2（アルゴリズム 1）において，(t, t') の代りに $(I(t), I(\tau))$ を考えれば図2はそのまま成り立ち，D 行列が定義できる．アルゴリズム2も同様に成り立つ．以上により図3の例では定義1を先にV実行し，定義2を後にV実行すると△印を付した1に相当する箇所において参照関係順序対が保存されず埋没定義が生ずる．

3. D 行列の操作による1重化された多重ループのベクトル化

3.1 文の並べかえによるデータ参照関係不適の解消と縮退部の検出[4]

データ参照関係順序対において，S実行で先行するデータ参照を含む文の文番号から，先行しないほうの文の文番号に矢が向いている有向グラフを考える．D 行列は実はこの有向グラフと等価であり，有向グラフ内に閉ループがあるかどうかを D 行列を用いて検査する．もし閉ループがあれば，そのループに属する各文はどのように並べかえても不適データ参照関係を解消できない．すなわちそのままではV実行できない．これは“循環”参照ともいうべき参照関係である．閉ループ中の各文は“縮退”させて1文として扱い（これを**縮退部**という），D 行列の下三角にはまったく1が立たないように文の並べかえを行い，縮退部以外はV実行する．縮退部に含まれる各文は，元来与えられた順にS実行すればよいが，しかしさらに式の代入[5]，配列変数のコピー等の方法により，この縮退部を解消してV実行またはそれをさらに縮小できる場合がある．

l を多重ループ本体の文の総数とすると，前述したように D 行列は $l \times l$ 行列である．D 行列は縮退があればサイズが $n \times n$ $(n \leqq l)$ に縮小する．文の並べ

かえによる不適データ参照関係の解消と縮退部の検出アルゴリズムは次のとおりである.

アルゴリズム 3 (D行列操作による文の並べかえ)

i は検査する D 行列の列番号 ($1 \leqq i < n$) で, D 行列の下三角の第1列から第 $i-1$ 列の部分はすでにゼロになっているものとする. すなわち, 文の並べかえ後の第1番目から第 $i-1$ 番目の文が, 並べかえ前のどの文であるかはすでにきまっているとする.

(i) 履歴を空にし, 第 i 列に相当する (並べかえ前の) 文の文番号 I を履歴に入れる.

(ii) D行列の第 i 列の第 $i+1$ 行から始めて, 第 n 行までに非零要素があるかどうか調べる. あれば (iii) へ. なければ第 i 列の検査終了で (vi) へ.

(iii) 第 j 行 ($i+1 \leqq j \leqq n$) に非零要素があるとき, 履歴に第 j 行に相当する文の文番号 J があるかどうかを調べる.

(iv) J が履歴にないときは, 履歴の最後に J をつけ加え, D 行列の第 i 列と第 j 列, および第 i 行と第 j 行をいれかえる (文 I と文 J をいれかえる). (ii) へ戻る.

(v) J が履歴にあるときは, 履歴の J と J より後にある文を縮退させる. すなわち, D 行列において, 縮退した文は, 縮退前の各文を表す行および列の値の行ごとおよび列ごとの論理和をとった一つの行および列として表される. このとき D 行列のサイズも小さくなる. 小さくなった D 行列を $n \times n$ として (ii) へ戻る.

(vi) D行列の第 i 列の第 $i+1$ 行から第 n 行まですべてがゼロにできたとき, 第 i 行に相当する文が, いれかえ後の第 i 番目の文である. 第 i 行に相当する文が, k 個の文が縮退した文ならば, いれかえ後のループ本体の第 i 番目の文として縮退した k 個の文を割り当てる. □

(vii) $i < n-1$ ならば, つづいて D 行列の第 $i+1$ 列を検査する.

以上の操作は要するに, 参照関係順序対を表す D 行列の非零要素を探すことにより, D 行列と等価な有向グラフを矢印の向きに辿りながら, 順に履歴として矢印の先の節点 (文番号) を記憶し, 履歴にすでに入っている文が再び現れるかどうか, つまり有向閉ループとなっているかどうか調べるわけである.

3.2 D行列表現によるデータ参照関係の解像力の不備

いま D 行列表現で 2 文のみが縮退し, それらを文1および文2とする. グラフ表示すると図4のように閉ループを作る (図4の①, ②はそれぞれ文1, 文2を表す). 文の左辺と右辺を区別すると, さらにこれには図5の(イ), (ロ)に示すように二つの場合がある. ただし図5で L は文の左辺, R は文の右辺を表し, 矢印は S 実行で先行するものから先行しないものへの参照関係の順序を表すことはいままでと同じである. (イ)の場合は真の意味で循環参照で V 実行できないが, (ロ)の場合はコピー用配列変数を1個図5(ハ)の斜線部分のように用意すれば上を向く矢印はなくなり, 参照関係順序対は保存されて V 実行できる.

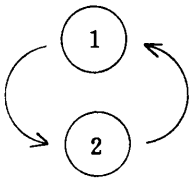

図 4 D 行列表現による循環参照 (2文が縮退の場合)
Fig. 4 Circular references by D-matrix representation.

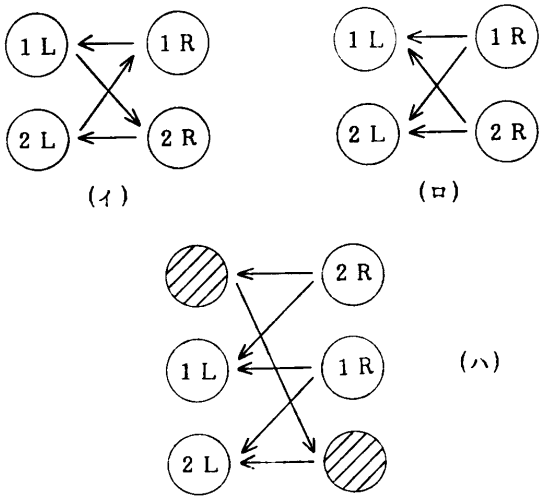

図 5 図4のさらに詳しい場合分け
Fig. 5 Further classification on Fig. 4.

このように特別の工夫により縮退部を解消したり, もしそれが多数の文を含むものならば縮小したりできる場合がある. この方法を多重ループの場合に拡張できるが, それに関しては別論文を準備中である.

4. 例による解説

以上述べてきた方法は, 本来ならばコンパイラの内部で展開されるべきものであるが, その考えの妥当性を検証するためソースプログラムのプログラム変換を

```
      REAL*8 A(10,50),B(10,20),C(20,40),X(0:10),Y(10)
      DO 100 I=1,10,1
         Y(I)=X(I-1)+1 ― ― ― ― ― ― ― ― ― ― ― ― ― 文1
        DO 200 J=I,2*I,1
           B(I,J)=A(I,J)+1 ― ― ― ― ― ― ― ― ― ― ― 文2
200     CONTINUE
         X(I)=Y(I)*5 ― ― ― ― ― ― ― ― ― ― ― ― ― ― 文3
        DO 300 K=18+2*I,8+2*I,-5
          DO 300 L=0,8,4
             A(I,K+L)=C(I+L,K) ― ― ― ― ― ― ― ― ― 文4
300     CONTINUE
100   CONTINUE
```

図 6 ソースプログラムの例
Fig. 6 An example of a source program.

```
      REAL*8 A(10,50),B(10,20),C(20,40),X(0:10),Y(10)
      INTEGER J0¥(65),I0¥(65)
      INTEGER L1¥(90),K1¥(90),I1¥(90)
      DATA J0¥ /
     &  1, 2* 2, 3, 4, 3, 4, 5, 6, 4, 5, 6, 7, 8, 5, 6, 7, 8, 9,10, 6,
     &  7, 8, 9,10,11,12, 7, 8, 9,10,11,12,13,14, 8, 9,10,11,12,13,14,
     & 15,16, 9,10,11,12,13,14,15,16,17,18,10,11,12,13,14,15,16,17,18,
     & 19,20/
      DATA I0¥ /
     &  2* 1, 3* 2, 4* 3, 5* 4, 6* 5, 7* 6, 8* 7, 9* 8,10* 9,11*10/
      DATA L1¥ /
     & 0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,
     & 8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,
     & 4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8,0,4,8/
      DATA K1¥ /
     &  3*20, 3*15, 3*10, 3*22, 3*17, 3*12, 3*24, 3*19, 3*14, 3*26,
     &  3*21, 3*16, 3*28, 3*23, 3*18, 3*30, 3*25, 3*20, 3*32, 3*27,
     &  3*22, 3*34, 3*29, 3*24, 3*36, 3*31, 3*26, 3*38, 3*33, 3*28/
      DATA I1¥ /
     &  9* 1, 9* 2, 9* 3, 9* 4, 9* 5, 9* 6, 9* 7, 9* 8, 9* 9, 9*10/
      DO 100 I=1,10,1
      Y(I)=X(I-1)+1
      X(I)=Y(I)*5
  100 CONTINUE
      DO 9901 I¥=1,65,1
      B(I0¥(I¥),J0¥(I¥))=A(I0¥(I¥),J0¥(I¥))+1
 9901 CONTINUE
      DO 9902 I¥=1,90,1
      A(I1¥(I¥),K1¥(I¥)+L1¥(I¥))=C(I1¥(I¥)+L1¥(I¥),K1¥(I¥))
 9902 CONTINUE
```

図 7 変換後の図6のプログラム
Fig. 7 The program given in Fig. 6 after having been transformed.

表 1 実行速度の測定値
Table 1 Measurement of execution speed.

機種	変換前		変換後
	S実行	V実行	V実行
VP 100	70.5209	76.1090	18.2273
S 810/20	93.27081	87.47914	15.56249

単位は μsec, 測定日はいずれも 1984年9月5日

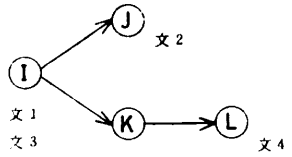

図 8 木構造で表した多重ループの構造
Fig. 8 The structure of a multiply nested loop represented by a tree.

行うプリプロセッサとして実現されている．すでに述べたようにインプリメンテーション上の詳細は別論文で論ずるが，本章では実例にもとづきプログラム変換の過程を示そう．**図6**のプログラムを考える．これを変換処理したものが**図7**のプログラムである．変換前と変換後の実行速度の測定値は**表1**のとおりで，われわれの変換により4倍から6倍の高速化が達成されている．

図6のプログラムで，文1と文3はラベル100のループに属し，文2と文4はそれぞれラベル200とラベル300のループに含まれている．したがって多重ループを表す木構造は**図8**のとおりで，節点を表す小

円の内部にはそのループの制御変数名が書いてある．

つぎに基本変数とそれによる制御変数の配列化であるが，文2の場合，基本変数は $t=1,2,\cdots,65$ の値をとり，それにより配列化された制御変数 $I(t)$ と $J(t)$ の値は，それぞれ図7のデータ文 DATA I 0¥ と DATA J 0¥ に t 値の昇順に示すとおりである．ただし，そのデータ文で，たとえば 10*9 は9という制御変数値が 10 個続くという意味である．文4の場合，基本変数は $t=1,2,\cdots,90$ の値をとり，それにより配列化された制御変数 $I(t)$, $K(t)$, $L(t)$ の値は，それぞれ図7のデータ文 DATA I 1¥, DATA K 1¥, DATA L 1¥ に示すとおりである．以上の情報など，構文解析時にループの構造情報，基本変数と配列化制御変数の情報，定義を含む複数出現の同一配列の情報を収集する．

データ参照関係の検査について次に述べる．2回以上少なくとも定義を含む参照関係で出現する配列は X, Y および A であるので，これらについて検査する．

（a） 配列 X について

文1で $X(I-1)$ を引用，文3で $X(I)$ を定義，したがってこの場合の Diophantus 方程式は

$$I-1=I'$$

で，文1と文3は同一ループに属しているので構造のない場合の扱いとなる．C 行列の非零要素は

$$(t\ (\text{引用}),\ t'\ (\text{定義}))=(2,1),\ (3,2),\ (4,3),\ \cdots,\ (10,9)$$

で，定義をタテ方向にとる C 行列のすべて上三角に落ち，つねに $t>t'$．すなわち図2の[A]の場合に該当していて，定義を行う文 s'（すなわち文3）を引用をする文 s（すなわち文1）より先行させなければ V 実行できない．$d_{31}=1$ とする．

（b） 配列 Y について

文1で $Y(I)$ を定義，文3で $Y(I)$ を引用，Diophantus 方程式は

$$I=I'.$$

これに対応する解 (t,t') は

$$(t\ (\text{引用}),\ t'\ (\text{定義}))=(1,1),\ (2,2),\ (3,3),\ \cdots,\ (10,10)$$

で C 行列は境界上にのみ1が立つ．つねに $t=t'$ かつ $s'\ (=1)<s\ (=3)$ で，図2の[A]の場合に該当し，文1を文3より先行させて $d_{13}=1$ とする．

（c） 配列 A について

文2で $A(I,J)$ を引用，文4で $A(I,K+L)$ を定義

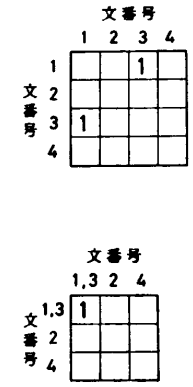

図 9 D 行列の操作
Fig. 9 Manipulation of D-matrices.

していて，Diophantus 方程式は

$$\begin{cases} I=I', \\ J=K'+L'. \end{cases}$$

構造のある多重ループの場合に該当する．上の連立方程式を数値的に解くと，各制御変数の上・下限の範囲で複数組の解 $(I,J;I',K',L')$ が存在するが，それを $(I(t),J(t);I(\tau),K(\tau),L(\tau))$ の形で満たす基本変数 t $(1\leqq t\leqq 65)$, τ $(1\leqq\tau\leqq 90)$ の組 (t,τ) が検索しても存在しない．したがってC行列の全要素がゼロで，文2と文4で現れる配列Aの要素の間には参照関係はない．

以上により図9（a）に示すようなD行列が得られた．これにアルゴリズム3を施すと図9（b）のようになり，文1と文3から成る縮退部が検出され，下三角の1がなくなり並べかえが終了する．これにより最終的な文の順序は，文1（S実行），文3（S実行），文2（V実行），文4（V実行）となる．この縮退部は3.2節で述べた真の意味での循環参照で，解消できない．以上の結果をまとめたのが図7に示した出力プログラムである．

5. むすび

本論文で述べた多重ループの自動ベクトル化の方法は，一応試験的にプリプロセッサの形で実現しており，現在さらに改良を加えつつある．しかしコンパイラの当然行う仕事と重なるところが多く，本来この方法はコンパイラの中に組み込まれるべきものである．

比較的すなおに書かれた実際のプログラム（構造力学計算）から切り出した多重ループなどについてテストしてみた．このプログラム群のうち比較的ベクトル化上条件のよいものでも企業提供の自動ベクトルコン

パイラではベクトル化率は約 85% どまりで，その同じプログラムに対し，プリプロセッサによる処理後ではベクトル化率はさらに約 10% 増加した．プログラム群全体としては，スカラ実行との処理速度比は処理前の平均 1.2〜1.5 倍から処理後には 3.4〜9.0 倍に向上した．ここに示した数値の開きは機種のちがいによる．

なお実際規模の適用例として，Hockney, Jesshope 共著[6]の図 5.14 (p. 243) 記載の FFT プログラムをそのまま用いた．その結果，1次元256点の場合，原プログラムをそのままV実行した場合に比し，プリプロセッサで処理後ではV実行の速さが3.69倍 (VP 100)，5.15倍(S 810/20) に向上し，また2次元 16×8 点では，これが4.32倍 (VP 100)，8.13倍 (S 810/20) となり，確実にわれわれの方法で高速化されることがわかった (1985年3月測定)．

本論文の1重ループ化の方法は変換により生ずる実行時オーバヘッドを伴うが，これを超えて有効となるためには，複数ループ間にまたがるデータ参照関係があり，かつ(a) ベクトル間接参照の対ベクトル直接参照の性能比が，1重ループ化されたときに生ずるような長いベクトル長に対してよい（おそくとも数倍程度），(b) 変換前多重ループに対し，変換後得られる1重ループの（V実行平均ベクトル長×ベクトル命令数）の比が大きい，ことである．(a)は load/store パイプの有効本数などハードウェア構成と関係がある．(b)は，もともとベクトル化率がそのままで十分高い多重ループは処理の対象とせず，たとえばやせた（文数の比較的少ない）最内側ループと太った（文数の多い）何重かの外側ループとの間にまたがって複雑なデータ参照関係がある場合，現在のコンパイラでは十分ベクトル化できないが，われわれの方法では外側ループを含め総じてV実行することにより，高速実行が可能であることを指している．

以上要するに，複雑な多重ループに対し，統一的に，一元的に自動ベクトル化する方法について述べた．ベクトル計算機の利点の一つは，その汎用機的な性格にあり，したがって次世代ベクトル計算機はわれわれの述べた諸点に留意してハードウェアを用意すれば，広範囲な科学技術計算の場においてハードウェアの高速性能を引き出せるプログラミングが可能となるものと思われる．

なお本論文ではわれわれの方法の原理的側面についておもに述べたが，そのインプリメンテーション上の技術的詳細については，「多重ループにおける縮退部の解消と縮小」および「プリプロセッサとしての実現」という表題で稿を改めて述べるよう準備中である．

参考文献

1) Mendez, R. H.: The Japanese Supercomputer Challenge, *SIAM News*, Vol. 17, No. 1, pp. 1 & 5; No. 2, pp. 3 & 16 (1984).
2) Kamiya, S., Isobe, F., Takashima, H. and Takiuchi, M.: Practical Vectorization Techniques for the "FACOM VP", in Mason, R. E. A. (ed.): *Information Processing 83*, pp. 389-394, Elsevier Science Publishers, North-Holland (1983).
3) 安村通晃，梅谷征雄，堀越 弥：自動ベクトルコンパイラにおける部分ベクトル化の方式，情報処理学会論文誌, Vol. 24, No. 1, pp. 15-21 (1983).
4) 二宮正和：京都大学大学院工学研究科修士課程情報工学専攻修士論文 (1984. 2).
5) Kuck, D. J., Kuhn, R. H., Leasure, B. and Wolfe, M.: The Structure of an Advanced Vectorizer for Pipelined Processors, COMPSAC 80, IEEE, pp. 709-715 (1980).
6) Hockney and Jesshope (奥川，黒住訳): 並列計算機，p. 243，共立出版，東京 (1984).

(昭和59年10月 5 日受付)
(昭和59年11月15日採録)