内蔵ベクトル演算機能のための自動ベクトルコンパイラ方式†

梅谷征雄†† 堀越彌††

FORTRAN などで書かれたユーザプログラム中の演算並列性を解析してベクトルプロセッサ用オブジェクトプログラムを生成するベクトルコンパイラはベクトルプロセッサのために必要となるソフトウェアのなかでも最も重要なものの一つである．ここではベクトルプロセッサ HITAC M-180/M-200H IAP (Integrated Array Processor) 用ベクトル・コンパイラの方式と特徴，効果について述べる．IAP 用ベクトルコンパイラの設計にあたっては，演算並列性の解析能力を高めるために配列指標・添字解析方式の強化を行ったほか，IAP の特徴的な演算レパートリである内積命令，積和命令などの活用方式と，一般命令との整合をとるためにベクトル命令のオペランド指定のために特別に必要となる制御テーブル領域の最小化に意を用いた．それらの工夫により，線形計算プログラムにおけるベクトル化可能 DO ループ数を従来の 20% から 50% に増加させることができ，またベクトル化によるオブジェクトプログラムサイズの増大も，線形計算プログラムの場合で 15% 以下に押えることができた．

1. まえがき

近年，半導体コストの低下とともに，多数の論理回路を使用して技術計算を高速に処理するベクトルプロセッサまたはアレイプロセッサとよぶ新型のプロセッサ群が台頭しつつある[1]．これらは従来のプロセッサと異なり，ベクトル命令と呼ばれる高い水準の命令を前提とし，ハードウェアにより内部的にこの命令を並列実行することによって高い性能を実現しているため，逐次処理を前提とした従来方式とは異なるプログラミング上の配慮が必要となってくる．しかし，技術計算では逐次計算を前提とするFORTRAN言語によるプログラミングが一般的であり，その膨大な財産と長年に渡るプログラミング習慣をいかにしてベクトルプロセッサに馴染ませるかが大きな課題となっている．これに対する一つの解答が，FORTRAN プログラム中の演算並列性を解析してベクトルプロセッサ用オブジェクトコードに自動的に展開するベクトルコンパイラであり，FORTRAN 言語仕様のベクトル向き拡張とともに，それを相補うものとして多くのベクトルプロセッサに採用されている[2)~4)]．これらのコンパイラでは，プログラム中の演算並列性の検出能力がその質を計る重要な尺度となる．

筆者らは従来どちらかというと高価なスーパーマシンに限られていたベクトルプロセッサの高速性を，汎用計算機でも実現するために，汎用計算機向きの内蔵ベクトル演算機能を開発し，HITAC M-180/M-200H IAP (IAP: Integrated Array Processor, 以下 M-180/200H IAP と略す) として実用化した．この内蔵ベクトル演算機能の実現に際しては，汎用計算機向きという用途を考えユーザプログラムからの透明性を重要な設計指針として自動ベクトルコンパイラに全面的に依存し，利用者は従来どおりFORTRANでプログラミングすることを前提とした．M-180/200H IAP では線形計算向きの内積，積和計算などの複合ベクトル演算命令を有すること，汎用命令との仕様の整合性を取るためにオペランドの指定用のために付加的な主記憶装置上の制御テーブルを必要とすること，および演算中間結果の保持のために主記憶装置上の中間ベクトル領域を必要とすることなどが特徴となっており，それだけ複雑な自動ベクトルコンパイラが要求された．本自動ベクトルコンパイラの開発に当たっては，演算並列性の解析能力を高めるためのより高度な配列指標解析および添字比較方式，内積演算，積和演算などのベクトル化方式，オペランド指定制御テーブルなどの付加記憶領域の最小化方式などの新しい方式の研究を行い，これを実用化した．本稿ではこれらの点を中心に述べる．

本論文では，2章で内蔵ベクトル演算方式と自動ベクトルコンパイラの概要を述べ，3章では今回新たに開発した自動ベクトルコンパイラの方式について詳し

† Automatic Vectorizing Compiler for Integrated Vector Arithmetic Facility by YUKIO UMETANI and HISASHI HORIKOSHI (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

く述べる．また，4章ではその効果を評価する．

2. 内蔵ベクトル演算機能と自動ベクトルコンパイラ

2.1 内蔵ベクトル演算機能の概要

内蔵ベクトル演算機能は処理装置に内蔵される科学計算用高速演算機構であり，**表1**に示す 28 種のベクトル命令を高速に処理することができる．ベクトル命令とは，1次元の指標（インデクス）により制御されるベクトルとよぶデータ集合間の加減乗除などの基本演算を行う命令であり，FORTRAN プログラムにおける DO ループなどの繰返し演算の高速処理に有効である．今回開発した内蔵ベクトル演算機能はとくに総和命令 (VSME, VSMD, VSMCE, VSMCD), 内積命令 (VIPE, VIPD, VIPCE, VIPCD), 積和命令 (VSMAE, VSMAD, VSMSE, VSMSD), 逐次計算命令 (VITRE, VITRD) のような複合命令を備えているのが特徴でありこれらを有効に利用する必要がある．

図1にベクトル命令の形式を示す．一般命令の形式と整合をとるために，ベクトル命令のオペランド指定部分を命令本体から切り離したいわゆるリスト形式をとっている．すなわち，命令の R3 フィールドで指定される汎用レジスタにより OAV (operand address vector) の先頭アドレスを指示する．OAV にはスカラならばオペランドアドレスが，ベクトルならばそのディスクリプタ VDT (vector descriptor table) のアドレスが置かれ，ベクトルの場合にはさらに VDT を経由してオペランド本体を指定する．VDT は一連のデータ集合の先頭アドレスと，隣り合う要素間のアドレス差分値を組にして保持するものであり，したがって，対象とするベクトルはアドレスが定差分のものに限られることになる．オペランドアドレスの変更は他の命令により OAV ないし VDT の内容を書き換えることにより行うことができる．このように命令語の

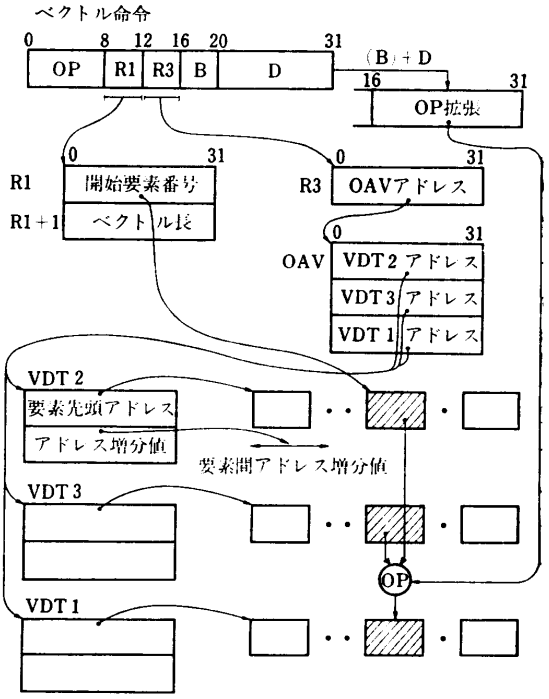

図 1　ベクトル命令形式
Fig. 1　Vector instruction format.

表 1　ベクトル命令一覧
Table 1　Summary table of vector instructions.

命令名称	略称 短精度	略称 倍精度	演算語の S/V 1	2	3	動作概要
Vector Move	VME	VMD	V	S/V		$Z_i \leftarrow X_i$
Vector Elementwise Complement	VECE	VECD	V	S/V		$Z_i \leftarrow -X_i$
Vector Elementwise Add	VEAE	VEAD	V	S/V	S/V	$Z_i \leftarrow X_i + Y_i$
Vector Elementwise Subtract	VESE	VESD	V	S/V	S/V	$Z_i \leftarrow X_i - Y_i$
Vector Elementwise Multiply	VEME	VEMD	V	S/V	S/V	$Z_i \leftarrow X_i * Y_i$
Vector Elementwise Divide	VEDE	VEDD	V	S/V	S/V	$Z_i \leftarrow X_i / Y_i$
Vector Element Sum	VSME	VSMD	S	S/V		FPR$\leftarrow$FPR$+\sum X_i$
Vector Element Sum with Complement	VSMCE	VSMCD	S	S/V		FPR$\leftarrow$FPR$-\sum X_i$
Vector Inner Product	VIPE	VIPD	S	S/V	S/V	FPR$\leftarrow$FPR$+\sum X_i * Y_i$
Vector Inner Product with Complement	VIPCE	VIPCD	S	S/V	S/V	FPR$\leftarrow$FPR$-\sum X_i * Y_i$
Scalar Multiply and Add	VSMAE	VSMAD	V	S	S/V	$Z_i \leftarrow Z_i + X * Y_i$
Scalar Multiply and Subtract	VSMSE	VSMSD	V	S	S/V	$Z_i \leftarrow Z_i - X * Y_i$
First Order Iteration	VITRE	VITRD	V	S/V	S/V	$Z_{i+1} \leftarrow X_i + Y_i * Z_i$
Convert Double to Single	VCVDE		V	S/V		Z_i (Single)$\leftarrow X_i$ (Double)
Convert Single to Double		VCVED	V	S/V		Z_i (Double)$\leftarrow X_i$ (Single)

ほかにオペランド指定のための付加的なテーブルが必要となる．また命令の R1 フィールドで指定される汎用レジスタの偶奇ペアの奇数番レジスタでベクトル長が決まるので，これを命令実行に先立ってセットしなければならない．偶数番レジスタは，命令の途中で割り込みが発生した場合の再開時に処理を開始すべき最初の要素番号を与える．(B2)+D2 フィールドでは行うべき演算の種類と各オペランドのスカラ，ベクトルなどの区別を指定する．

2.2 自動ベクトルコンパイラの機能

FORTRAN 用の自動ベクトルコンパイラは，FORTRAN プログラムの最内側 DO ループ部分に対してベクトル演算の適合性を判定し，適合する場合にはベクトル命令からなるオブジェクトプログラムを，適合しない場合には，従来どおりスカラ命令からなるオブジェクトプログラムを生成する．ベクトル化に適合するループに対してベクトル命令を生成するか否かをコンパイラ時のオプション指定により制御することもできる．**図2** に FORTRAN のソースコードと，コンパイラの生成するオブジェクト・コードの一例を示す．図2左下がベクトル命令を使用しない従来のオブジェクトコード，右下がベクトル命令を含むオブジェクトコードである．

図2からわかるように，DO 10 で指定される繰返し演算を，ベクトルコンパイラは VEME, VEAE からなる二つのベクトル命令に展開する．VEME 命令はベクトル (X(I))I=1,N と (Y(I))=I=1,N の要素ごとの乗算を行い，結果を記憶装置上の中間ベクトル (T(I))I=1,N におき，続く VEAE 命令によって (Z(I))I=1,N と (T(I))I=1,N の加算を行い，結果を再び (Z(I))I=1,N に返す．左下のオブジェクトに示す本来の演算ループを各ベクトル命令内の単一演算ループの列に変換するわけであり，ベクトルコンパイラは演算順序の組み替えを行うことになる．また，従来，浮動小数点レジスタを介して伝えられている演算の中間結果は記憶装置上の中間ベクトル領域に置かれる．

コンパイラの能力を示す DO ループに置けるベクトル演算の適合条件は，今回開発した内蔵ベクトル演算機能用ベクトルコンパイラでは**表2** のようになっている．まず，DO ループ内のベクトル化の対象となる文種類は，制御構造と演算種の簡素化をはかるために代入文と CONTINUE 文のみに限定した．次に代入文内の演算種類については，ベクトル命令の種類の制約から四則演算と主要基本外部関数に限定し，ユーザ定義関数などは許容していない．変数，配列のデータ形式については，やはり命令種類の制約から添字計算を除いて，実数型単精度，倍精度のみを許容する．また，配列添字に関しては，命令形式の制約から最内側ループ内で値の変化しない定添字，またはループの進行につれて定値で変化する線形添字に限定する．この条件の判定には後でのべる指標・添字解析が必要になる．最後にベクトル演算の特性に由来するものとして，データ参照関係に関する制約がある．これは“ベクトル命令の使用によりループ内のデータ間参照順序が変化してはならない”という制約条件であり，その意味を**図3** により説明する．

ソースプログラム

```
CARD    ISN     SOURCE STATEMENT
   1    00001   DIMENSION X(100),Y(100),Z(100)
   2    00002   DO 10 I=1,N
   3    00003 10 Z(I)=Z(I)+X(I)*Y(I)
   4    00004   STOP
   5    00005   END
```

オブジェクトプログラム

NOIAP

```
          LTR    2, 2
          BC     13,=#100001
100002    LR     2, 9
          LR     10, 3
          LR     11, 2
   10     LE     2,X
          ME     2,Y
          AE     2,Z
          STE    2,Z
          AR     11, 9
          BCT    10,=#10
10000+    LR     8, 6
          AR     8, 7
100001    L      15,=V(F#RCON)
          BAL    14,  52( 0,15)
```

IAP

```
100000    L      10,=F'1'
          L      2,N
          SR     2,10
          AR     2,10
          BC     13,#100001
   10     SR     0, 0
          L      1,N
          LA     15,=0A001
          VEME   0,15
          LA     15,=0A002
          VEAE   0,15
100004    L      11,N
          AR     11,10
100001    L      15,=V(F#RCON)
          BAL    14,  52( 0,15)
```

図 2 ソースプログラムとオブジェクトプログラム例
Fig. 2 An example of source and object program generated by IAP FORTRAN compiler.

図3の二つの DO ループ，DO 10 と DO 20 はいずれもベクトル演算に適合しない添字関係を含むものである．DO ループ中で2回参照が行われる配列Aの添字関係が問題であり，ベクトル化は，本来の DO ループを右に示すような文ごとの二つのループにわけるのと等価であることから，共通要素たとえば A(2) の参照順序がベクトル化時に変化することになる．すなわち，DO 10 においては，本来 s_2 にて引用される A(2) 値は s_1 にて定義される以前の値であるべきであるが，ベクトル化時にはそれが逆転する．ベクトル演算に適合するデータ参照関係の条件として，配列データについては，定義，引用位置とそれぞれの添字値との

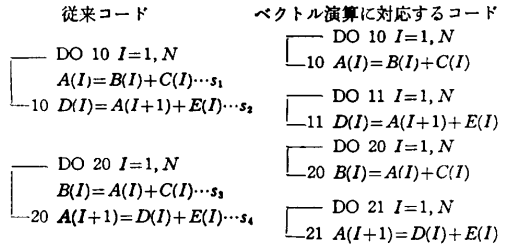

図 3 ベクトル化に適合しないデータ参照関係を含むコードの例

Fig. 3 Code examples which include data reference relations unsuitable for vectorization.

関係が問題であり，コンパイラはこのために指標・添字解析に基づく添字比較を行う必要がある．以上述べた適合条件の判定をコンパイラはベクトルコードの生成に先立って行う．

2.3 自動ベクトルコンパイラの構成

図 4 にFORTRAN用自動ベクトルコンパイラの構成を示す．この図はコンパイラのモジュール階層を示しており，ベクトル化機能に関連した部分をクローズアップしてある．処理の流れもほぼこの階層トリーをたどって得られる．

ベクトル化処理はおもに構文解析部とストレージ割り付け部の間で行われる．すなわち，構文解析と同時に最内側 DO ループに対する文種類の判定を行った後，それに適合した DO ループについては，ここでその他の判定と中間語レベルでのベクトル化コードへの変更を行う．すなわち，配列指標解析，データ型判定，添字解析による添字形式判定，データ参照関係判定を順次行った後，OAV, VDT への値設定コードならびにベクトルコードを含む中間テキストを生成する．ストレージ割り付け部では，通常の変数，配列とともに，OAV, VDT，中間ベクトルなどにアドレスを割り付けるコード最適化部，オブジェクト生成部を経由してベクトル化中間テキストをベクトル命令列に変換する．最後のメッセージ処理部ではエラーメッセージとともに，各最内側 DO ループのベクトル化状況を出力する．

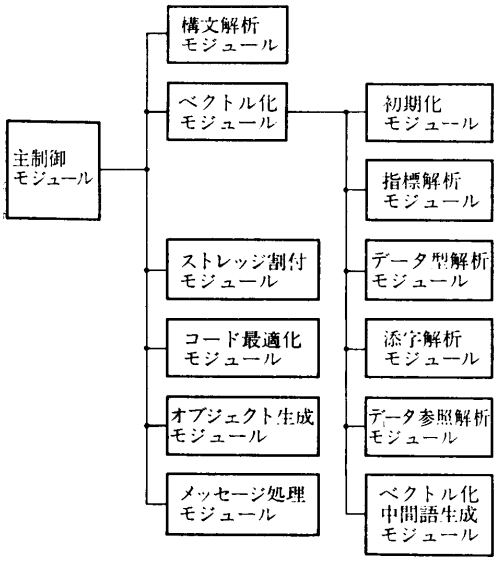

図 4 内蔵ベクトル演算機構用自動ベクトル化 FORTRAN コンパイラの構成

Fig. 4 Module structure of the automatic vectorization FORTRAN compiler designed for the integrated vector arithmetic feature.

3. 自動ベクトルコンパイラの方式

ベクトルコンパイラの目的は，プログラム中の DO ループ演算のできるだけ多くの部分をベクトル命令コードに変換することにある．表2に示したベクトル化適合条件中，配列添字形式とデータ参照関係によるベクトル化の制約はより高度のコンパイラの指標・添字解析方式，および添字比較方式により明確に判定され，従来不明とされたより多くの部分がベクトル命令コードに落とせるようになるので，この方式の改善はコンパイラの質を高める上で非常に重要である．一方，他のベクトルプロセッサとは違った内蔵ベクトル演算機能特有の解決を要する問題として，線形計算の高速化を目的として付加された内積命令，積和命令の利用方法，および一般命令との仕様の整合性から派生する OAV, VDT, 中間ベクトルなどの管理方法が上げられる．前者については，内積命令に対応する DO ループは一般的なベクトル化適合条件に当てはまらないデータ参照関係を有するため，自動ベクトル化のためには特例的な配慮が必要である．また後者については，実行時のプログラムサイズと管理オーバヘッド

表 2 内蔵ベクトル演算機能用自動ベクトル・コンパイラにおける，DO ループ内のベクトル化条件

Table 2 Conditions to be satisfied for a DO loop to be vectorized by the automatic vectorization compiler of integrated vector arithmetic facility.

判定条件の種類	ベクトル化適合条件
(1) 文の種類	代入文，CONTINUE 文
(2) 演算の種類	四則演算および主要基本外部関数
(3) データ形式（添字計算を除く）	実数型の単精度または倍精度
(4) 配列添字形式	定添字または線形添字
(5) データ参照関係	添字関係がベクトル化に整合していること（3章参照）

を小さく押さえるため，記憶領域の共有化を徹底する必要がある．本章では，この3点に焦点を合わせて自動ベクトルコンパイラの方式上の特徴を述べることにする．

3.1 配列指標解析と添字比較方式

配列指標解析は，VDT (vector descriptor table) の内容，すなわちベクトルの先頭アドレスとアドレス増分値の決定に必要であり，また図3に示した配列間のデータ参照解析に要求される添字比較の基礎情報を与える．

ところで，従来の指標解析方式は，I を DO 文で指定される指標変数 (DO 指標変数)，C を定数とするとき，各次元の添字表現が，C, I, C*I, I±C で表記される単純な範囲に限定しているが[2),9)]，実プログラムにおいては指標操作の複雑さから，**図5**に示す IND のようにループ内で IND=IND+2 といった形式で再修正される再帰型指標変数や，I1, I2, INDJ のようにループ内で再定義される再定義指標変数のような種々の形式が用いられる．また，FORTRAN 77 言語などでは添字部に一般の式が書けるように文法が拡張されていることから，種々の指標変数の解析方式を確立し，またそれに応じて添字比較方式を拡張する必要がある．以下に今回開発した方式を示す．

(1) 指標変数と添字式の表現形式

図5に示すような配列の添字比較において，再帰型指標変数 IND や，再定義指標変数 I1, I2 などを統一的に扱うためには，従来の DO 指標変数を基準とする添字表記方法は適切とはいえない．そこで指標変数とこれらによる添字式を（初期値，増分値，終了値）の三つの組で表現することとする．I1, I2, INDJ のようなループ中での置き換えを考慮すると，初期値，増分値，終了値は一般にループ中で値をかえない変数（ループ相対定数）を含む式となる．このような表現形式はまた，VDT に設定すべきベクトルアドレスを明示するためにも必要である．

```
  ISN      SOURCE STATEMENT
 00001     DIMENSION A(100),B(100),C(100)
 00002     DO 10 I=1,N
 00003     I1=N1-I
 00004     I2=I1-1
 00005     INDJ=IND+J
 00006     A(I1)=A(I2)
 00007     B(IND)=C(INDJ)+3(IND-1)
 00008     IND=IND+2
 00009  10 CONTINUE
 00010     STOP
 00011     END
```

図5 種々の指標変数例
Fig. 5 Examples of various index variables.

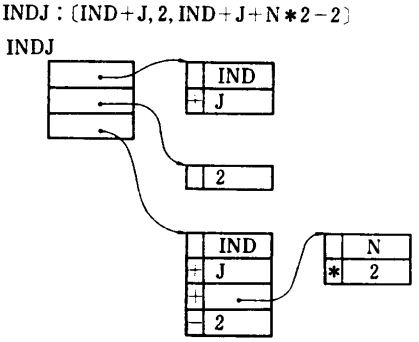

図6 指標添字の表現形式
Fig. 6 Standard representation scheme of index variables and a subscript expression with them.

図6に，図5の INDJ に対する表示形式と記憶装置上の表現形式を示す．初期値，増分値，終了値の各式を記憶装置上では和項部と積項部にわけて保持し，三つの和項部の先頭アドレスをさらに上位のテーブルにて保持する形式を取る．ここで各式が積和で表現されるもののみを解析の対象とし，括弧は展開により除去する．またループ内で定義される変数にはその定義式を代入することにより標準化する．さらに，各式において，和項の配置順序および積項内の定数，変数の配置順序は，定数，変数に与えた順序数に基づき辞書式に定める．積項内の複数の定数は計算により一つにまとめる．この表現形式により表示の一意性が保証される．このような式の表現形式を“標準表現”とよぶことにする．

図5における，他の指標変数と添字式の表現形式は次のとおりである．

I : (1, 1, N)
I1 : (N1−1, −1, −N+N1)
I2 : (N1−2, −1, −N+N1−1)
IND : (IND, 2, IND+N×2−2)
IND−1 : (IND−1, 2, IND+N×2−3)

(2) 指標変数と添字式の標準化手続き

上記の表現をえるための手続きのあらましは次のとおりである．

(a) 定数およびループ相対定数を検出する．ループ相対定数にはループ中で定義されるものを含み，指標変数の各式と同様，標準表現にて記憶する．

(b) 指標変数を検出する．指標変数は，DO 指標変数と再帰型指標変数をまず検出し，

次いでループ中で定義される指標変数を上から順に検出する．この際，定義文の右辺に対して，定数ないしループ相対定数を係数とする検出済指標変数の一次式という制約が課せられる．

（c） 添字式を標準化する．配列の各次元の添字式が標準化されるためには，それが，定数またはループ相対定数のみからなる積和式かつ指標変数の一次式でなければならない．前者に対しては増分値をゼロとして標準化する．

（3） 標準表現間の演算

上記の添字式同士を比較するために必要な標準表現間の演算を定義する．E1 と E2 を二つの標準表現とする．

（a） 加減算 (E1±E2), 乗算 (E1×E2)：通常の積和式間の演算として定義する．

（b） 除算 (E1/E2)：E2 が定数の場合にのみ定義し，E1 の各定数係数を E2 で割る．

（c） 大小比較 (E1>E2, E1=E2, E1<E2)：E1 と E2 の変数を含む項が完全に一致し，定数部のみに差のある場合に定数部により比較する．

（d） 最大公約数 (E1∩E2)：E1, E2 ともに定数の場合にのみ通常の方法で定義する．

（4） 添字比較方式

上記の標準表現に基づき従来の添字比較方式を拡張する．

添字比較は，ループ内の2か所以上で参照され，定義と定義，あるいは定義と引用の関係にある同一配列の配列要素の間で行う必要がある[2]．それらの間のデータ参照関係がベクトル化に適合するための一般的な条件は以下に述べるとおりであり[2]，それを判定する操作が添字比較である．

ある配列 A の参照対を取り上げ，文番号 s における l 回目のループ時の配列添字を $X_s(l)$，したがって参照する配列要素を $A(X_s(l))$ とし，他方の文番号 t で参照する配列要素を $A(X_t(l))$ とすると，いかなるループ指標値 l, l' に対しても $X_s(l) \neq X_t(l')$ であるか，$X_s(l)=X_t(l')$ の場合には，$s \leqq t$ かつ $l \leqq l'$，あるいは $s \geqq t$ かつ $l>l'$ であることがベクトル化を許容する条件である．

文献[2]などにおいては，X_s, X_t はすべて DO 指標変数の一次式であったことから増分値が共通であり，上記の判定を行うためには一次式の係数と定数同士の比較を行えばよかった．しかし，今回拡張添字表現を用いて増分値の異なる添字同士を比較することからその拡張を行った．その方式の概要は次のようである．比較対象とする次元 M の配列 A の1対の要素参照の一方を $A(is1, is2, \cdots, ism)$，他方を $A(i11, i12, \cdots, i1m)$ とする．$isj, i1j$ を（初期値，増分値，終了値）の三つ組により $(B0j, D0j, F0j)$，$(B1j, D1j, F1j)$ と表現すると，値0から増分値1でループ回数 $N-1$ に至る仮想的なループ指標 l を導入することにより，$i0j=B0j+D0j\times l$, $i1j=I1j+D1j\times l$ と表現できる．

判定方法を付録に示す．その大筋は，まず $P1=\{(l, l') \mid B01+D01\times l=B11+D11\times l'\}$ を構成し，その要素を次々と先の次元のテストに掛けることにより，最終的に $P=\bigcap_j Pj$ を構成し，P の要素にたいして大小判定を行う．j を進めるに従い，$\bigcap_j Pj$ は面集合（2次元）から線集合（1次元），点集合（0次元），空集合へと変化する．空集合に落ちる場合はそこで共通要素をもたなくなるので判定を打ち切る．また特定の次元で $\bigcap_j Pj$ を再限定できない場合にはそのままにして先の次元に処理を任せる．

この添字比較方式により図5の例における A(I1) と A(I2) では付録に示した判定方法 case 1.1 により P は1次元集合となり，そこに属するすべての (l, l') に対して $l>l'$ が判定される．また，B(IND) と B(IND−1) では step 1.1.3 で P が空集合と判定される．その結果，データ参照関係がベクトル化に適合していることが判定される．

3.2 再帰的演算のベクトル化方式

行列 A, B 間の乗算

```
DO 10 I=1, N
DO 10 J=1, N
DO 10 K=1, N
10 C(I, J)=C(I, J)+A(I, K)×B(K, J)
```

における最内側ループの内積演算をベクトル化の対象とすると，代入文両辺にあらわれる $C(I, J)$ のデータ参照関係が問題となる．これについて，添字値を等しくするループ指標の集合 P を求めると，I, J がともに最内側ループで値をかえないことから，

$$P=\{(l, l') \mid l \in (1, N), l' \in (1, N)\}$$

となり，P の要素における l と l' の大小順序は確定せず，ベクトル化に適合しない．内積，総和，逐次計算などの再帰的演算に対しては，データ参照に関して特例的な配慮が必要となる．

内積演算に対するベクトル化適合条件を次のように定めることによりそのベクトル化を実現することがで

きる.

(1) 同一代入文の左辺，右辺間でスカラ変数によるデータ参照関係を有すること．スカラ変数は単純変数でも上記の $C(I,J)$ のような配列要素でもよい．スカラ変数は右辺に唯一回出現するものとする.

(2) 右辺の式は二つの項からなり，一つは当該スカラ変数，他は二つの式の積であること．すなわち，F1,F2 を式とするとき S±F1*F2 あるいは F1*F2 +S の形であるものとする.

(3) 当該スカラ変数は，DO ループ中の他の文に出現しないこと.

条件 (1)，(3)はデータ参照解析により，(2)は特殊な構文解析により判定する.

ベクトル総和演算に対しては，(2)にてスカラ変数が最外側和項にあるという条件が必要であり，他の制約は取り払われる．逐次計算に対しては，(1)にてスカラ変数は，左辺と右辺の添字差が1のベクトル配列の条件に置き換えられ，条件(2)はベクトル総和と同じとなる．また，条件(3)は不要となる．**図7**に，内積演算のベクトル化オブジェクト例を示す．VIPE が内積命令であり，直前の LA は OAV のアドレス設定命令，それに先立つ LE は $C(I,J)$ の初期値を浮動小数点レジスタにおく命令，VIPE に続く STE は結果の $C(I,J)$ への格納命令である.

```
 ISN      SOURCE STATEMENT
00001     DIMENSION A(100,100),B(100,100),C(100,100)
00002     DO 10 I=1,N
00003     DO 10 J=1,N
00004     DO 10 K=1,N
00005  10 C(I,J)=C(I,J)+A(K,I)*B(K,J)
00006     STOP
00007     END
```

オブジェクトコード

```
          100005   L      3,@010
                   A      3,=F'4'
                   L      2,@004
                   A      2,@003
                   L      4,@001
                   L      8,=I0002
                   LR     9, 2
                   L      10,@009
00004     100006   LTR    4, 4
                   BC     13,#100007
          100008   LA     14,A
                   ST     14,=VD001
                   LA     14,B
                   ST     14,=VD002
00005        10    SR     0, 0
                   L      1,N
                   SDR    0, 0
                   LE     0,C
                   LA     15,=0A001
                   VIPE   0,15
                   STE    0,C
          100007   AR     9, 5
                   AR     10, 5
                   BCT    8,#100006
```

図 7 内積演算のベクトル化オブジェクトコード
Fig. 7 Vectorized object codes of inner product operations.

3.3 付加記憶領域の最小化方式

ベクトル化に伴う付加記憶領域，OAV (operand address vector), VDT (vector descriptor table), 中間ベクトル領域のうち，VDT と中間ベクトル領域は種々の領域共有化方式の採用により，所要量の削減をはかることができる．VDT の共有化はまたその内容更新のための命令を削減する効果があり，性能向上にも寄与する．本節では，VDT と中間ベクトル領域の領域共有制御方式を述べる.

(1) VDT の領域共有制御

VDT は，ベクトル要素の先頭アドレスとアドレス増分値を保持するテーブルであり，共有制御はループ内共有化とループ間共有化に分けられる．ループ内共有化は，ループ内での同一配列の同一添字による複数の参照に対して単一の VDT を使おうとするものであり，VDT がデータに付随することからくる当然の帰結といえる．その VDT の内容が外側ループの指標により変化する場合には，VDT 更新の手間を節約する効果も期待できる.

一方，ループ間共有化は，ループの入り口でアドレス設定が必要な VDT にたいして，ループ間にわたり領域を共有化するものである．この場合も，先頭アドレスと増分値のいずれか一方が定数の場合には，同じ定数値のものを共有させることによりアドレス設定の手間を節約できる．さらに先頭アドレスと増分値がともに定数の場合には，当然その定数の組をループ間で共有させるようにする．以上の共有化制御を実現するためのコンパイラの作業テーブルの構造を**図8**に示す.

共有制御は図8に示すとおり，1本の VDTCTBL チェインとその各エントリから派生する VDTTBL チェインにより行う．VDTTBL チェインの各エントリはベクトル命令の参照する VDT を代表するもので，最終的にはこれを用いて VDT が生成される．VDTCTBL チェインは，T1, T2, … で示すあらかじめ決めた数の中間ベクトル領域用エントリと，Si で示すベクトル化される単純変数のエントリ（ソースコードでは単純な DO ループ内の中間変数であっても，その DO ループがベクトル化されると，単純変数もベクトル化する必要が生じ

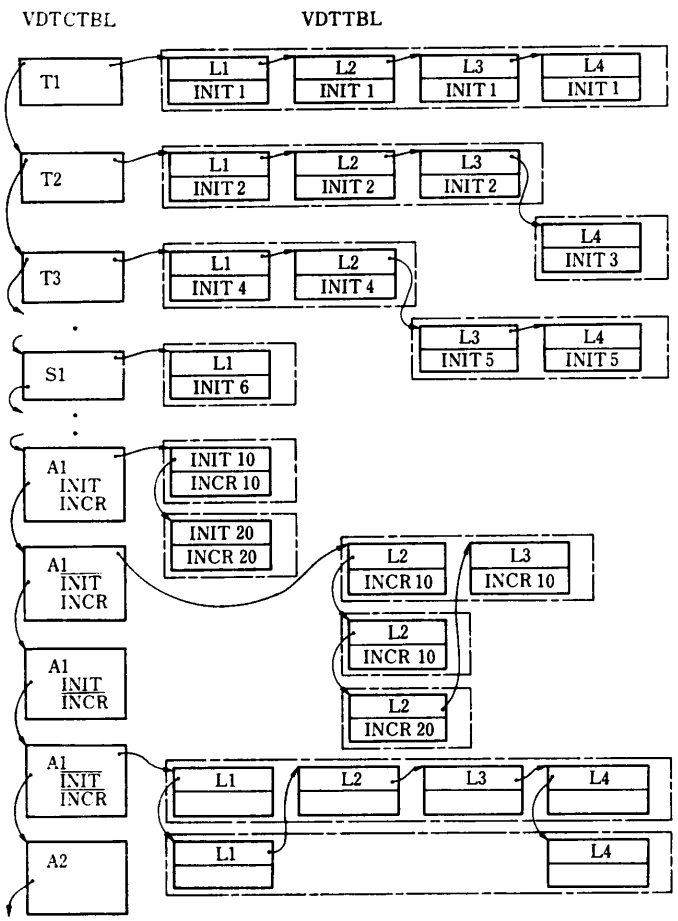

図 8 VDT の領域共有化の作業テーブル構造
Fig. 8 Structure of work tables used to minimize the storage areas of VDT's.

る）および各配列の先頭アドレスと増分値の定数/非定数の種別に対応して各配列 A*i* ごとの四つのエントリからなる．図にて INIT, $\overline{\text{INIT}}$ は先頭アドレスが定数，非定数，INCR, $\overline{\text{INCR}}$ は増分値の定数，非定数をそれぞれ示す．中間ベクトル領域 T1, T2, … に関しては，中間変数の参照要求のあった DO ループごとに VDTTBL エントリを用意してループ番号 L*i* とともに記録する．S*i* については，参照要求のあったループに対して一つずつ VDTTBL エントリを用意する．その後，各ループの中間ベクトル必要数とループ長を勘案して各ループにおける中間ベクトルの先頭アドレス INIT*i* が決められるので，各 VDTTBL チェイン上で先頭アドレスのおなじものに同一の VDT アドレスを割り付け，領域の共有化を図る．図8にて一点鎖線で囲まれる VDTTBL エントリから生成される VDT は同一のアドレスに作成される．

先頭アドレスと増分値がともに定数の配列 (INIT, INCR) に対しては，アドレス設定を省くために参照ループにかかわらず定数値の組だけ VDT を用意しアドレスを割り付ける．先頭アドレスが非定数，増分値が定数の各配列 ($\overline{\text{INIT}}$, INCR) については，参照のあるループごとにループ番号 L*i* と増分値 INCR*i* を組にした VDT を用意し，ループ間にわたる同一増分値の VDT を同一アドレスに割り付ける．同一増分値の VDT が同一ループ内で複数回参照される場合には，先頭アドレスも同一である場合を除いてループ内共用はできないので異なるアドレスを割り付ける．(INIT, $\overline{\text{INCR}}$) の各配列に対しても同様である．最後に ($\overline{\text{INIT}}$, $\overline{\text{INCR}}$) の各配列に対しては，参照ループごとに必要数の VDT を用意し，ループ間で共有化を図る．

以上の方式により，VDT 領域の共有化を制御する．

（2） 中間ベクトル領域の共有制御

中間ベクトル領域の共有制御は次の各レベルで行う．

（a） 文内共有制御：式中の演算中間結果の受け渡しに関してレジスタの割当て制御と同様の方式により，本数の最小化を図る．

（b） 文間共有制御：中間ベクトルは文間に渡ることがないので，文内共有制御により自動的に文間共有制御も実現される．

（c） ループ間共有制御：おのおのの中間ベクトル領域はループ回数によって変化するので，ループごとに中間ベクトルの領域を調整し総量の最小化を図る．コンパイル時にループ回数が不明な場合は，ループ中で指標が動く配列の次元に着目し，その配列宣言の最大値により一つの中間ベクトル領域の大きさを決定する．これも不可能な場合はユーザ指定値ないしコンパイラ標準値を用いる．あるループ内での中間ベクトル領域の総量はユーザ指定の上限値による規制も行う．すなわち，総量が上限値を越える場合には，自動的に DO ループのベクトル化を抑止している．

ループ間共有制御の一例を**図 9** に示す．

（d） サブルーチン間共有制御：中間ベクトルをコモン領域にとることにより実現する．

4. コンパイラの方式の評価

前章に述べた諸方式の効果を，2本の線形計算ライ

ブラリプログラムにより評価する．プログラムAは，LU 分解による連立一次方程式求解プログラム，Bは QR・逆反復法による固有値・固有ベクトル求解プログラムである．

4.1 ベクトル化能力評価

表3に，両プログラムのベクトル化状況を，再内側 DO ループ数を単位として分類した結果を示す．これらはともにループ演算主体のプログラムであり，再内側ループ部分の処理時間が全処理時間の 90% 以上を占めている．

プログラムAは8個の再内側 DO ループ部分を含み，うち五つがベクトル化可能である．ベクトル化可能ループ中，3.1 節の拡張された配列指標解析・添字比較処理が有効であったものは一つ，3.2 節の再起的演算ないし同種の積和演算処理が有効であったものは二つとの結果が得られた．プログラムBについては，52個中46個の最内側 DO ループがベクトル化され，内拡張指標添字処理の有効なものは 10 個，再帰的演算処理などの有効なものは 15 個となっている．

指標添字処理の有効性はプログラムの表記法に依存するのでより多くの評価を積む必要があるが，再帰的演算の出現は計算アルゴリズムに依存しており線形計算ではとくにこれらのベクトル化が有効であることがわかる．

表3 線形計算プログラムのベクトル化状況
Table 3 Evaluation of vectorization ability of the compiler.

DO ループの性質と対応するコンパイラ技術	プログラム A	プログラム B
拡張添字比較方式の導入によりベクトル化された DO ループ	1	10
従来技術でもベクトル化可能な DO ループ	4	36
内積，積和の再帰演算命令にベクトル化された DO ループ*	(2)	(15)
非再帰演算としてベクトル化された DO ループ*	(3)	(31)
ベクトル化されたループの小計	5	46
今回ベクトル化されなかったループ数	3	6
全最内側ループ数	8	52

* この分類は上の添字比較方式による分類と重複している．

4.2 記憶領域の評価

両プログラムの記憶領域評価を，オブジェクトプログラムサイズ，VDT 領域共有効果にわけて**表4**に示す．オブジェクトプログラムサイズは，ベクトル化の指定を行う場合と行わない従来の場合を対比して内訳を示す．オブジェクトプログラムサイズについて，プログラムAはベクトル化指定時に OAV，VDT，データ領域などが若干増えるが，命令コード部が少し減るために合計では大差がない．プログラムBはベクトル命令が多いために OAV 領域が増えるほか，VDT，OAV セットアップ命令の増加により命令部が全体的にふくらみ，合計で約 10% のサイズ増となっている．このほかに中間ベクトル用コモン領域が 3,000B 必要である．これらのプログラムはいずれもライブラリプログラムの範疇に属するもので実際のデータと作業領域は一般にはメインプログラムから引数で渡される．通常のプログラムではデータ領域の比率はさらに高いのが普通なので，プログラムBのベク

表4 線形計算プログラムの記憶領域評価
Table 4 Evaluation of the storage minimization capability of the compiler.

比較項目		内訳		プログラムA ベクトル化時	プログラムA 従来	プログラムB ベクトル化時	プログラムB 従来
オブジェクトプログラムサイズ		命令領域		1,992B	2,086B	14,424B	14,152B
		OAV		56		1,408	
		VDT		32		240	
		データ領域		136	120	488	428
		中間ベクトル領域		(0)		(3,000)	
		その他		524	516	1,448	1,420
		合計		2,740B	2,722B	18,008	16,000
VDT領域共有効果	配列ベクトル	VDT数	(a)	9		127	
		領域割当て数	(b)	4		13	
		共有効果	(a/b)	2.3		9.8	
	中間ベクトル	VDT数	(c)	0		46	
		領域割当て数	(d)	0		17	
		共有効果	(c/d)			2.7	
中間ベクトル領域共有効果		要求中間ベクトル量	(e)	0		23,000B	
		領域割当て量	(f)	0		3,000B	
		共有効果	(e/f)			7.7	

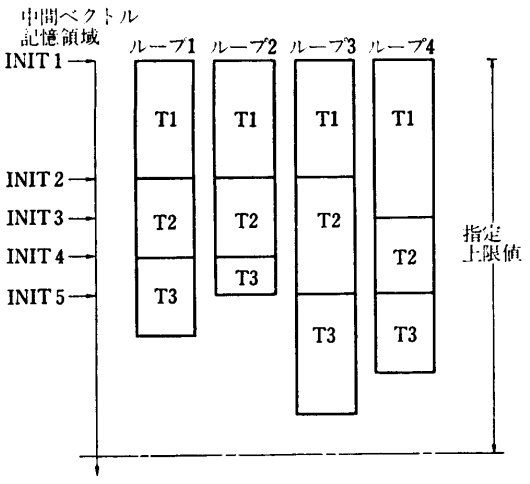

図 9 中間ベクトル領域のループ間共有制御
Fig. 9 Minimization control scheme of temporary vector area.

トル化によるこの程度のサイズ増はそれほど問題ないと思われる.

次に VDT 領域共有効果を配列ベクトルと中間ベクトルに分け OAV より指定される VDT 数と実際に領域を割り当てた VDT 数の比で評価する. 表 4 に示すように配列ベクトルに関してはプログラムAで 2.3, プログラムBで 9.8 とかなりの共有効果が上がっている. 一方, 中間ベクトルに関しては, プログラムAはそれを必要とせず, プログラムBで 2.7 となった. 中間ベクトル用 VDT の共有化率が低いのは, 中間ベクトルの割当て制御 (図 9) により先頭アドレスの異なるケースが多いので共有化が阻害されるためと思われる. しかし全般的には高い効果を上げているといえる.

最後に中間ベクトル領域の共有化効果を, 各ループで中間データの受け渡しに使われるベクトルサイズの総和 (e) と割当てサイズ (f) の比で評価するとプログラムBで 7.7 との高い値が得られた. プログラムBの結果はサブルーチン間共有効果を考慮していないので, 複数ルーチンにわたる大規模プログラムでは, より効果が大きいものと予想される.

5. むすび

以上, M-180/200H 内蔵ベクトル演算機能用 FORTRAN コンパイラの概要と方式上の特徴, ならびにその効果について述べた.

当コンパイラは, 基本機能の充実と線形計算への適用を図るためデータフロー解析に重点をおいているが, 今後さらにベクトル化能力を強化するためには, これらの機能をいっそう拡充すると同時に, 条件文を含む DO ループをベクトル化するための制御フロー解析や関数計算のベクトル化などを行ってゆく必要があると考える.

謝辞 最後に, 本コンパイラの開発にあたってご指導ならびにご協力いただいた, (株)日立製作所中央研究所 村田技師長, (株)日立製作所ソフトウェア工場 高須部長, 松本部長, 大木主任技師, 高貫主任技師, 同神奈川工場 小高主任技師, ならびに筑波大学電子情報工学系 中田教授に厚く感謝の意を表します.

参考文献

1) 小高俊彦, 他: 超高速演算の動向, 情報処理, Vol. 21, No. 9, pp. 927-937 (Sep. 1980).
2) Presberg, D. L. et al.: The Paralyser: IVTRAN's Parallelism Analyser and Synthesizer, *ACM SIGPLAN Notices*, Vol. 10, No. 3, pp. 9-16 (Mar. 1975).
3) Wedel, D.: FORTRAN for the Texas Instruments ASC System, *ACM SIGPLAN Notices*, pp. 119-132 (Mar. 1975).
4) CRC: CRAY-1 FORTRAN (CFT) 説明書, SM-007 (Apr. 1980).
5) 日立製作所: HITAC M-180/M-200H 内蔵アレイプロセッサ解説書, 8080-2-041-10 (Feb. 1980).
6) 小高俊彦, 他: HITAC M-180 内蔵アレイプロセッサ, 日立評論, Vol. 60, No. 6, pp. 53-58 (Jun. 1978).
7) 河辺 峻, 他: HITAC M-200H 内蔵アレイプロセッサ, 電子通信学会電算機研究会, EC80-79, pp. 67-75 (Mar. 1981).
8) 中沢喜三郎, 他: HITAC M-200H 汎用高速処理装置, 日立評論, Vol. 61, No. 12, pp. 7-12 (Dec. 1979).
9) Lamport, L.: Parallel Exection of DO Loops, *C. ACM*, Vol. 17, No. 2, pp. 83-93 (Feb. 1974).
10) 堀越 彌, 他: 汎用計算機のための内蔵ベクトル演算方式, 情報処理学会論文誌, Vol. 24, No. 2, pp. 191-199 (1983).

付録 拡張添字比較方式

initialization SW←3

loop: Do for $j=1,2,\cdots,M$ (P の計算をする)

case 1: SW=3 の場合 (第1回目の P の作成, または, 既作成 $\bigcap_j P_j$ が2次元集合の場合)

case 1.1: $D_j * D'_j \neq 0$ (ただし, D_j, D'_j はともに定数とする)

step 1.1.1: D_j, D'_j の最大公約数 g を求め

る．

step 1.1.2: $c \leftarrow D'_j/g$, $c' \leftarrow D'_j/g$

step 1.1.3: $B_j + D_j * l = B'_j + D'_j * l'$ を満たす．1組の (l, l') を求め，これを (l_1, l'_1) とする．この組が存在しない場合は，SW←0 として go to exit ($\bigcap_j P_j = \phi$).

step 1.1.4: ループ回数 N に対して

$0 \leqq l_1 + c * k \leqq N$

$0 \leqq l'_1 + c' * k \leqq N$

を同時に満たす k の最小値，最大値を求め k_l, k_h とする（(l_1, l'_1) を基準にして，i_j は c 回ごと，i'_j は c' 回ごとに，$i_j = i'_j$ となる）．

step 1.1.5: $l_{\min} \leftarrow l_1 + c * k_l$, $l_{\max} \leftarrow l_1 + c * k_h$

$l'_{\min} \leftarrow l'_1 + c' * k_l$, $l'_{\max} \leftarrow l'_1 + c' * k_h$

step 1.1.6: SW←2 (case 1.1 の終り $\bigcap_j P_j$ は1次元集合または点)

case 1.2: $D_j = 0, D'_j \neq 0$ の場合

step 1.2.1: $l_{\min} \leftarrow 0, l_{\max} \leftarrow N-1$

$l'_{\min} = l'_{\max} \leftarrow (B_j - B'_j)/D'_j$

step 1.2.2: $l'_{\min} (= l'_{\max})$ が整数，かつ $0 \leqq l'_{\min} < N$ の場合，$c \leftarrow 1, c' \leftarrow 0$, SW←2 (case 1.2 の終り，1次元集合) そうでない場合は，SW←0 として go to exit ($\bigcap_j P_j = \phi$)

case 1.3: $D_j \neq 0, D'_j = 0$ の場合

step 1.3.1: $l'_{\min} \leftarrow 0, l'_{\max} \leftarrow N-1$

$l_{\min} = l_{\max} \leftarrow (B'_j - B_j)/D_j$

step 1.3.2: $l_{\min} (= l_{\max})$ が整数，かつ $0 \leqq l_{\min} < N$ の場合，$c \leftarrow 0, c' \leftarrow 1$, SW←2 (case 1.3 の終り) そうでない場合，SW←0 として go to exit ($\bigcap_j P_j = \phi$)

case 1.4: $D_j = 0, D'_j = 0$ の場合

$B_j = B'_j$ の場合 continue $B_j \neq B'_j$ の場合は，SW←0 として go to exit ($\bigcap_j P_j = \phi$)

case 2: SW=2 の場合（既作成の $\bigcap_j P_j$ が一次元集合）．

step 2.1: $D_j * c \neq D'_j * c'$ の場合

go to step 2.3 (添字増分値が以前の次元でのものと一致しない)

step 2.2: $B_j + D_j * l_{\min} = B'_j + D'_j * l'_{\min}$ の場合 continue

step 2.3: $B_j + D_j * (l_{\min} + c * k_0) = B'_j + D'_j * (l'_{\min} + c' * k_0)$, $0 \leqq k_0 < N$ なる k_0 を探す．そのような k_0 が存在する場合には，SW2←1, $l \leftarrow l_{\min} + c * k_0, l' \leftarrow l'_{\min} + c' * k_0$ として continue，存在しない場合には，SW←0 として go to exit ($\bigcap_j P_j = \phi$).

case 3: SW=1 の場合（既作成の $\bigcap_j P_j$ が，0次元集合，すなわち，1点のみ含む場合）

step 3.1: $B_j + D_j * l = B'_j + D'_j * l'$ の場合 continue そうでない場合，SW←0 として go to exit ($\bigcap_j P_j = \phi$).

exit: (添字関係の判定処理)

付表1に示すように，SW の値と，$l_{\min}, l_{\max}, l'_{\min}, l'_{\max}$ を使って，l と l' の大小関係を決める．

付表 1 添字比較の判定規準
A Table 1 Decision rules to determine which loop indices is larger when the corresponding suffixes are identical.

SW の値		
3	(2次元集合)	$(l, l') \in P$ なる l と l' の大小関係不定
2	(1次元集合)	
	$l_{\min} \leqq l'_{\min}$, $l_{\max} < l'_{\max}$ or $l_{\min} < l'_{\min}$, $l_{\max} \leqq l'_{\max}$	$l < l'$ for all $(l, l') \in P$
	$l_{\min} = l'_{\min}$, $l_{\max} = l'_{\max}$	$l = l'$ for all $(l, l') \in P$
	$l_{\min} \geqq l'_{\min}$, $l_{\max} > l'_{\max}$ or $l_{\min} > l'_{\min}$, $l_{\max} \geqq l'_{\max}$	$l > l'$ for all $(l, l') \in P$
	それ以外の場合	$(l, l') \in P$ なる l と l' の大小関係不定
1	(1点集合)	l と l' の大小関係により決める
0	(空集合)	データ参照関係なし

(昭和57年7月13日受付)
(昭和57年10月4日採録)