Vol. 27 No. 4 情報処理学会論文誌 Apr. 1986

ショートノート

自動ベクトル化コンパイラにおける添字比較方式についての提案†

丸島敏一†† 村岡洋一††

通常のFORTRANプログラムから並列性を見出す自動ベクトル化コンパイラにおいて，添字比較方式はその主要な部分を成すものである．しかしながら，従来の添字比較方式はパイプラインプロセッサを主な対象としており，マルチプロセッサのためのベクトル化には不十分であった．今回これを改良して，ベクトル化可能となる範囲の拡大を可能とし，またそれ以外の部分についてもデータ参照関係を式で表すことにより，データ制御の際の助けとした．

1. はじめに

現在市場にある並列処理計算機はパイプライン方式が主流であるが，さらに高速な処理をねらうにはマルチプロセッサ方式の採用が必須となる．しかし，パイプライン方式の並列処理計算機用に開発された，FORTRAN自動ベクトル化コンパイラでは，ある配列の定義参照関係について，一部でもベクトル化不可能な部分があるとその文はすべてスカラ命令に展開されてしまい，マルチプロセッサ方式のものに適用するには不十分である．本論文においては，従来の自動ベクトル化コンパイラでベクトル化可否の判断に採用している添字比較方式を拡張した，新しい添字比較方式を提案する．これは，簡単に言えば1組の2文間の定義参照関係について，すべての添字値の中でベクトル化可能となる範囲とベクトル化不可能となる範囲との区分けを行い，ベクトル化不可能となる範囲においては，どれだけ後にその更新されたデータを参照するかという情報を与えるものである．このような機能を持つことによって，マルチプロセッサ方式における並列処理範囲を拡大できる．

ある配列が定義もしくは参照される場合，ベクトル化可否の判定をする際考慮しなければならないのは，(定義→参照)，(参照→定義)，(定義→定義) の2出現，または3回以上出現する場合にはその組み合わせについてである．ベクトル化可能なデータ参照とは，ベクトル実行によってもスカラ実行によっても2出現の順序が変わらないことを言う (**図1**)．また，同一文で参照・定義をしている場合には参照後定義していると考えられるので，(参照→定義) の特別な形とみなせる．

これまでに開発された自動ベクトル化コンパイラでは，ベクトル演算は，配列要素すべてについて外見上同時に行うという原則があった．つまり，演算上の機構についてはいっさい関知せず，演算に必要な配列要素をすべてまとめて演算部に受け渡し，その結果として配列要素をすべてまとめて受け取るという考え方である．そのため，

```
loop 1 : do i =10 to 100
           A(i)=A(i-5) * B(i)
         end do
```

のようなループはベクトル化できないことになる．ところが，マルチプロセッサでは**図2**のような並列処理が可能である．このように，ループ回数を限定してベクトル化できることを分割ベクトル化可能と呼ぶ．すなわち，マルチプロセッサ方式のための自動ベクトル化コンパイラでは，ループ全体のベクトル化のみならず，ループをいくつかのベクトル化可能な集合に分割することも必要となる．また，この時の各々の集合をベクトル処理単位と呼ぶ．さらに，

```
loop 2 : do i=1 to 100
           B(i)       =A(2*i+500)
           A(7*i+35)=C(i)*C(i+1)
         end do
```

という場合には，$i=1$〜93では分割ベクトル化可能で

† A Proposal of Subscript Comparative Method for Automatic Vector Compilers by TOSHIKAZU MARUSHIMA and YOICHI MURAOKA (Department of Electronics and Communication Engineering, School of Science and Engineering, Waseda University).

†† 早稲田大学理工学部電子通信学科

あるが，$A(700)$ という要素に対して $i=95$ で定義したものを $i=100$ で参照するということが行われるため，これまでのコンパイラではたった一つの例外のために全体がベクトル化不可能となってしまう．マルチプロセッサでは，$i=1$〜93 についてベクトル実行を施し，それ以外の部分については文の入れ替えもしくは，分割ベクトル化を施す等の工夫により並列処理する．

なお，これらの考え方はパイプライン方式の並列計算機にも応用することができる．例えば loop 1 の場合，**図 3** に示すような機構を設けて演算結果を入力側にフィードバックすると，事実上ベクトル処理できる．つまり，この場合のベクトル処理単位はパイプライン・フィードバック段数にほかならない．

	ベクトル化可能	ベクトル化不可能
参照	= A(i)	= A(i)
↓	⋮	⋮
定義	A(i-2) =	A(i+2) =
定義	A(i) =	A(i) =
↓	⋮	⋮
参照	= A(i-2)	= A(i+2)
定義	A(i) =	A(i) =
↓	⋮	⋮
定義	A(i-2) =	A(i+2) =

図 1 データ参照関係とベクトル化可否の例
（制御変数 i は昇順とする）
Fig. 1 Examples of the data reference relation and vectorization.

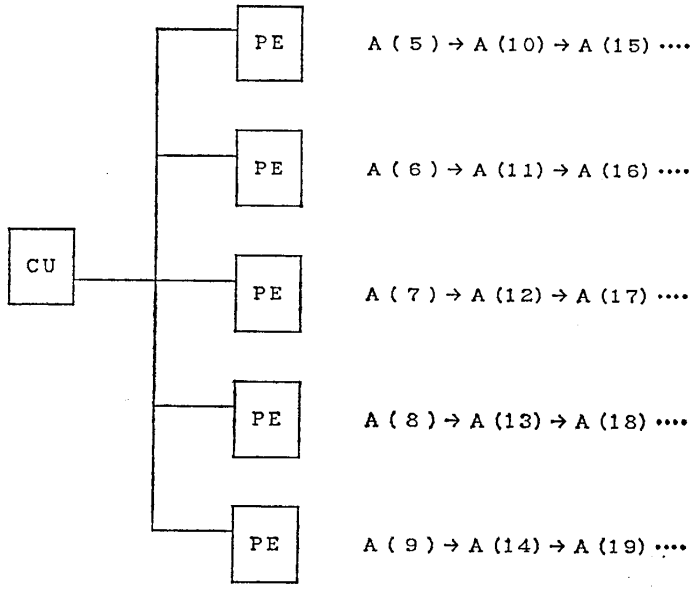

図 2 マルチプロセッサにおける並列処理の一例
Fig. 2 Parallel processing by multiprocessors.

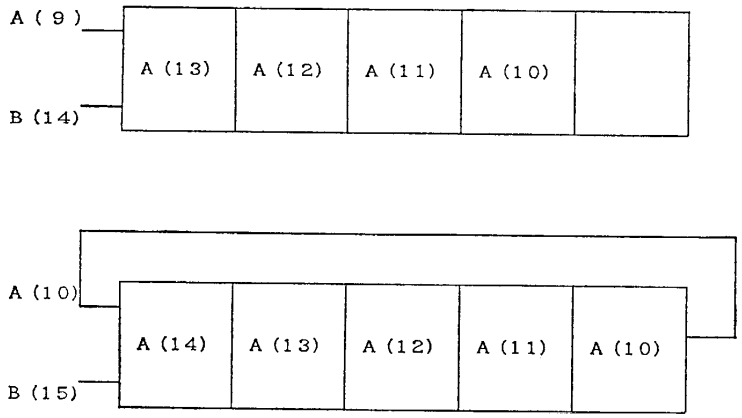

図 3 パイプラインにおけるフィードバックの例
Fig. 3 Feedback by pipelining.

2. 添字比較方式

2.1 従来の添字比較方式

従来の添字比較方式[1]の原理について簡単に説明する．まず，次元 m の配列 A の一対の要素参照の一方を $A(f_1, f_2, \cdots, f_m)$，もう一方を，$A(f_1', f_2', \cdots, f_m')$ とし，添字 f_j, f_j' $(j=1\sim m)$ を（初期値，増分値，終了値）の三つ組により，(B_j, D_j, F_j)，(B_j', D_j', F_j') で表す．ただし，この三つ組はすべて定数で表されるものとする．すると，標準化ループ指標 $l=0\sim N-1$ によって，$f_j=B_j+D_j*l$, $f_j'=B_j'+D_j'*l$ と表される．これを用いて，先行する文番号における配列要素に対するループ指標 l と，後続する文番号における l' について，$P_j=\{(l, l') | B_j+D_j*l=B_j'+D_j'*l'\}$ なる集合の j による積，つまり，$P=\bigcap_j P_j$ を最終的に求める．この際，$j=1$ から順番に $j=m$ まで添字比較方式が適用されていくごとに，求める集合 P は2次元集合→1次元集合→1点集合→空集合というように限定されていく．この判定基準を**表 1**[2]に示す．ただし，s, s' は文番号を表し，l_{min}, l_{max}, l'_{min}, l'_{max} は，$P=\{(l, l')\}$ の最小要素 (l_{min}, l'_{min}) 及び，最大要素 (l_{max}, l'_{max}) を形成するものである．

2.2 添字比較方式の拡張

前出の表1において，ベクトル化不可能となっているものについて，タイプ別に高速化を図る．その方法は次のとおりである．

表1 添字比較の判定基準
Table 1 Index variable comparison rules.

集合 P	データ参照関係[2]		ベクトル化可否	タイプ
2次元集合	$(l,l')\in P$ なる l と l' の大小関係不定		不可	I
1次元集合	$l_{min}\leqq l'_{min}$, $l_{max}<l'_{max}$	$l<l'$ for all $(l,l')\in P$	$s\leqq s'$ ⇒可能	—
	$l_{min}<l'_{min}$, $l_{max}\leqq l'_{max}$		$s>s'$ ⇒不可	II
	$l_{min}=l'_{min}$, $l_{max}=l'_{max}$	$l=l'$ for all $(l,l')\in P$	可能	—
	$l_{min}\geqq l'_{min}$, $l_{max}>l'_{max}$	$l>l'$ for all $(l,l')\in P$	$s\geqq s'$ ⇒可能	—
	$l_{min}>l'_{min}$, $l_{max}\geqq l'_{max}$		$s<s'$ ⇒不可	II
	上記以外の場合	$(l,l')\in P$ なる l と l' の大小関係不定	不可	III
1点集合	$(l,l')\in P$ なる l と l' は一組のみ		$(l-l')*(s-s')\geqq 0$ ⇒可能	—
			$(l-l')*(s-s')<0$ ⇒不可	IV
空集合	データ参照関係なし		可能	—

タイプI： $D_j=D_j'=0$ かつ $B_j=C_j'$ $(j=1,\cdots,m)$, つまり，その配列の添字は定数で表され，全く同じ要素を表していることが，アルゴリズムより分かる．もし，総和・総積などの特別な形でなければ，他の配列に展開する．この結果，ベクトル化できない場合は，タイプIIと同様に扱う．

タイプII：文の入れ替え，もしくは，後述する方法によりベクトル処理単位を求め，分割ベクトル化を施す．

タイプIII：図4, 5[1]を比較すると分かるように，$f_j(l)$ と $f_j'(l')$ が交わっているため，その前後で $f_j(l)$ と $f_j'(l')$ の大小関係が逆転している．つまり，$0\leqq l\leqq l_s$ においてベクトル化可能ならば，$l_s<l<N$ ではベクトル化不可能であり，また，その逆も成り立つ（ただし，$f_j(l_s)=f_j'(l_s)$）．よって，このような l_s を求め，この点を分割点として，$0\leqq l\leqq l_s$, $l_s\leqq l<N$ の二つの範囲について添字の比較を行えば，配列要素の途中でデータ参照関係が入れ替わっている場合のベクトル化の可否が，それぞれの l の範囲を限定して決定される．これによってベクトル化不可能と判断された部分については，再びタイプ別に分けて処理する．

タイプIV：得られた (l, l') 以外の部分についてベクトル実行し，(l, l') に関する部分には，何らかの方法でデータをフィードバックする．もしくは，全範囲についてベクトル実行し，後から補正計算式を付加することにより，結果の妥当性を保持する．

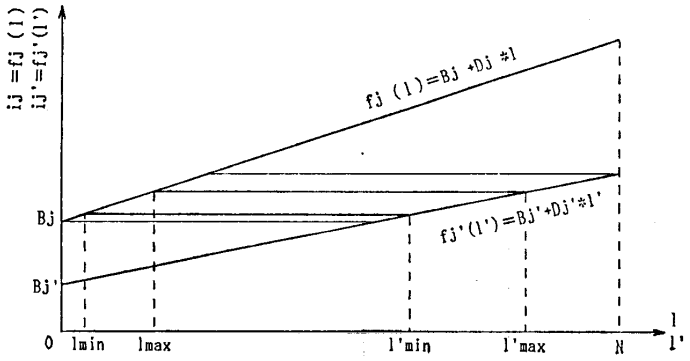

図4 l, l' の大小関係が確定する場合[1]
Fig. 4 Case of definite relation of l and l'.

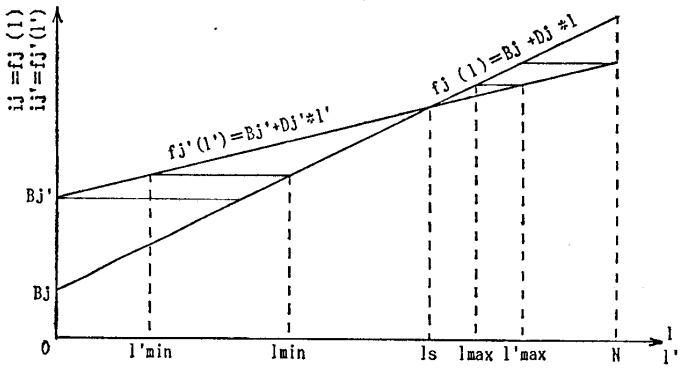

図5 l, l' の大小関係が不定となる場合[1]
Fig. 5 Case of indefinite relation of l and l'.

ところで，タイプIIの分割ベクトル化におけるベクトル処理単位であるが，これは次のように求めることができる．まず，D_j, D_j' の最大公約数を g とすると，ある (l, l') を基準として f_j は $C(=D_j'/g)$ 回ごとに，また f_j' は $C'(=D_j/g)$ 回ごとに $f_j=f_j'$ と

なる．これを利用すれば，l, l' は $l_{min} \leqq l_{min} + kC \leqq l_{max}$ を満たす k の関数として次のように表される．

$$l = l_{min} + kC$$
$$l' = l_{min}' + kC'$$

よって，ベクトル処理単位は，

$$(l_{min} - l_{min}') + k(C - C')$$

で与えられる．

このように，従来の添字比較方式を改良して利用することにより，ベクトル化可能となる範囲とベクトル化不可能となる範囲を区分けし，さらにベクトル化不可能な部分についてベクトル処理単位を k の関数として表すことにより，分割ベクトル化が可能となった．

3. 具体例による解説

3.1 逐次演算を含むループ

まず，例として loop 3 のループを考える．

```
loop 3 : do i=0 to 50
           A(12*i+30, 8*i+20)
             =A(9*i+159, 6*i+100)*B(i)
         end do
```

右辺の参照時ループ回数を l，左辺の定義時ループ回数を l' とすると，添字比較法より，集合 $P \ni (l, l')$ は1次元集合であり，

$$C = 4,\ C' = 3$$
$$(l_{min},\ l'_{min}) = (0,\ 10)$$
$$(l_{max},\ l'_{max}) = (48,\ 46)$$

となることが分かる．これに対して，表1より判定を行うと，“$(l, l') \in P$ なる l と l' の大小関係不定”ということになるので，従来の方法ではベクトル化不可能と決定される．そこで，このデータ参照関係が逆転する $l = l_s$ なる点において分割して考える．すると，$(l_s, l_s) = \{(l, l') | l = l'\}$ より，

$$l_{min} + kC = l'_{min} + kC'$$
$$k = 10$$
$$\therefore\ l_s = 40$$

が得られるので，loop 3 は分割ベクトル化され，

$i = 0 \sim 40 \cdots\cdots l \leqq l' \Rightarrow$ ベクトル化可能

$i = 41 \sim 50 \cdots\cdots l > l' \Rightarrow$ ベクトル化不可能

となることが分かる．つまり，従来ベクトル化不可能とされたループにおいて，大部分がベクトル化可能であったことが判明した．また，$i = 41 \sim 50$ の部分について新たに添字比較法を適用すると，

$$(l_{min},\ l'_{min}) = (44,\ 43)$$

と修正されるので，ベクトル処理単位として，

$$(l_{min} - l'_{min}) + k(C - C') = 1 + k\ (k = 0, 1)$$

が得られる．

次に，loop 3 の A が3次元だった場合として loop 4 の例を考えよう．

```
loop 4 : do i=0 to 50
           A(12*i+30, 8*i+20, 12*i+600)
             =A(9*i+150, 6*i+100, 24*i)
              *B(i)
         end do
```

これは，1次元目と2次元目の前の例と一致している．このループに関して添字比較方式により得られるのは，(l, l') は以下の一点のみであるということである．

$$(l, l') = (48, 46)$$

これに対しては，まず，$i = 0 \sim 46$ までをベクトル実行し，さらに残りの部分をベクトル実行してもよいし，または，とりあえずこのループを全範囲についてベクトル実行し，その後で補正計算として次のような式を付加することも考えられる．

$$A(12*48+30,\ 8*48+20,\ 12*48+600) = A(9*46+30,\ 6*46+100,\ 24*64)*B(46)$$

上式は，本来 $i = 46$ において定義された値を $i = 48$ において参照しなければならないのに対して，ベクトル実行では $i = 46$ において再定義される前の値を $i = 48$ において参照しているため，これに対する補正を行っている．

3.2 簡単な多重ループへの適用

ここでは，これまでのアルゴリズムの応用として，多重ループにこれを適用する方法について考える．ただし，対象となる配列は各次元に含まれる変数が一つまでの場合に限られる．これは二つ以上の変数が一つの次元に含まれている場合には，その添字増分値が一定にならず，各添字の増分値は定数で与えられるという前提に矛盾するからである．

さて，一般に多重ループのベクトル化では，ループの一重化という方法がとられるが，ここでもこの方法を用いることにする．例として，loop 5 のループを実行するとしよう．

```
loop 5 : do i=4 to 7
           do j=3 to 6
             do k=2 to 6
               A(k, j, i)
                 =A(k-1, j-2, i-3)
                  *B(i, j, k)
```

```
        end do
      end do
    end do
```

このように，それぞれの次元が別々の制御変数によって与えられる場合，まず，それぞれの次元での比較を独立に行い，もし，一か所でも循環参照をしない次元があれば，その配列はベクトル化可能であると判断される．もし，すべての次元で循環参照が行われているならば，それぞれのループの深さを考慮して，それぞれのループ回数をブロックとして考えることにより，その参照関係を求めることができる．Loop 5 では，配列 A について，全次元において循環参照が行われている．つまり，ベクトル処理単位は，i について 3，j について 2，k について 1 となっている．ループの深さを考慮すれば，k について 1 というのは一重ループの場合と変わらないが，j についてはそのカウントのそれぞれについて $k=2$〜6 という 5 回のループが入るので，最終的に j の値が二つ増えるのは，一重化されたループ回数において 5×2 回ごとということになる．同様に，i については，(4×5)×3 回ごとに 3 増える．つまり，全体として，

$$1\times1+5\times2+(4\times5)\times3=71$$

が，ベクトル処理単位となる．これにより，loop 5 は，配列要素 71 個までを同時にベクトル実行できることが分かる．

4. むすび

このように，従来の添字比較方式を改良して利用することによって，ベクトル化可否をループ回数の範囲を限定して求めることができた．また，さらにデータ参照関係を k の関数として，これを明確な形で表した．しかし，この方法には非線形添字に対する手段や，複雑な多重ループに対する方策が欠けているため，さらに考察を加え，マルチプロセッサ用自動ベクトル化コンパイラの実現のために助けとしたい．

参考文献

1) Takanuki, R., Nakata, I. and Umetani, Y. : Some Compiling Algorithms for Array Processor, Proc. of 3rd USA-Japan Computer Conf., pp. 273-279 (1978).
2) 梅谷征雄，堀越 彌：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，情報処理学会論文誌，Vol. 24, No. 2, pp. 238-248 (1983).
3) Muraoka, Y. : Parallelism Exposure and Exploitation in Programs, Ph. D. Thesis, Univ. of Illinois at Urbana-Champaign, Department of Computer Science, Report (1971).

(昭和 60 年 9 月 27 日受付)
(昭和 61 年 1 月 17 日採録)

丸島 敏一
昭和 38 年生．昭和 61 年早稲田大学理工学部電子通信学科卒業．現在，同大学大学院修士課程在学中．並列処理，コンピュータアーキテクチャに興味を持っている．

村岡 洋一（正会員）
昭和 17 年生．昭和 40 年早稲田大学理工学部電子通信学科卒業．昭和 41 年〜46 年イリノイ大学でイリアックⅣのソフトウェア開発に従事．昭和 46 年同大学博士課程修了．昭和 46 年〜59 年電電公社電気通信研究所．この間 DIPS の研究・開発に従事．昭和 60 年早稲田大学理工学部教授．Ph. D. コンピュータアーキテクチャ，マンマシンインタフェース等に興味を持つ．電子通信学会，ACM，IEEE 各会員．