プログラミング言語 14－2
（１９８７．１２．１８）

スーパーコンピュータのための言語処理システム

堀田 耕一郎，鈴木 清文，神谷 幸男

富士通株式会社

コンパイラによる自動ベクトル化の処理が軽減できるため、ベクトル処理を行うスーパーコンピュータ向けに、言語仕様の中に配列記述を導入する試みがなされてきた。しかし、配列記述を行うために発生する問題点も多く、そのための最適化を行わないと実用上満足できる言語であるとは言い難い。この報告では、既に開発済の配列記述を持つ言語(FORTRAN 77/VPベクトル拡張言語）での経験を元に、配列記述仕様を持つ言語で必要な最適化、及び最適化制御行について述べる。ここで述べた機能の追加によって、配列記述を行ったプログラムの性能を従来のプログラミングによるものと同等以上にすることができる。

Language processing system for Supercomputers

Koh-ichiro Hotta, Kiyofumi Suzuki, Sachio Kamiya

FUJITSU Limited.
140 Miyamoto, Numazu, Shizuoka, 410-03 Japan

The array notation have been added for some vector machines to avoid automatic vectorization by the compiler. But there are some problems caused by array notation. So, we cannot be satisfied without new optimization for the array notation. We have already developed the language processing system 'FORTRAN 77/VP Vector Extended feature', which has array notation. We show the optimization technique which is implemented after surveying the real program and our planning optimization controling line which should be added to the standard of the language which has array notation. Adding these optimizing facilities to the language processor system, we can get as high performance as programs written by the current FORTRAN 77.

1. はじめに

ベクトル（並列）プロセッサ向けに、ベクトルデータおよびそれに対する演算を直接に記述する配列記述を、言語仕様の中に導入する試みが今までなされてきた。配列記述は、自動ベクトル化に変わるものとして、更に、記述性が向上する言語仕様として、ベクトルプロセッサ向けに望ましい仕様であるという見方がされている。

当社は、既に配列記述仕様を持つ『FORTRAN 77/VP ベクトル拡張言語』を開発した。配列記述に対する問題点は、配列記述のための最適化の機能を追加しないと、FORTRAN 77と同等、あるいはそれ以上の性能が出てこないことである。本稿では、配列記述を導入する際にコンパイラに必要な最適化機能について報告し、配列記述に伴って取り入れられることが望ましい仕様、最適化制御行を提案する。

2. 配列記述の問題点

配列の全要素、あるいは必要ないくつかの要素をまとめて扱えるようにするための配列記述方法を、以下のように定義する。

$$A(N_1:N_2:N_3)$$

A を一次元の配列とすると、$A(N_1:N_2:N_3)$ は、一連の配列要素 $A(N_1)$,$A(N_1+N_3)$,$A(N_1+2N_3)$,・・を要素とするベクトルを表すものとする。すなわち、添字がN_1からN_2までN_3づつ増えるような配列要素をひとまとめに表すものである。

配列記述を言語仕様に導入することにより、コンパイラの自動ベクトル化能力に関わらず、プログラマが分かっているベクトル演算は全てベクトル命令に変換されるため、コンパイラの自動ベクトル化を補う意味での、単にベクトル化率を向上させるためのプログラムチューニングは不要となる。また、プログラマは配列演算をDOループに展開する必要がないので、ソースプログラムの記述性も向上する。

配列記述されたプログラムをFORTRAN 77のDOループで表現すると、一つの文が一つのDOループに対応する。

```
      DO 10 I=1,N
       T1   =(A(I)+B(I))/2.0
       C(I)=T1*T1*3.14
   10 CONTINUE
```

(1)従来のプログラミング

```
      DO 10 I=1,N,VL *1
       vr(1:VL)=
     &         (A(I:I+VL-1)+B(I:I+VL-1))/2.0
       C(I:I+VL-1)=
     &         vr(1:VL)*vr(1:VL)*3.14
   10 CONTINUE
```

(2)自動ベクトル化による実行イメージ

図1　FORTRAN 77/VP での自動ベクトル化

```
      T1(1:N)=(A(1:N)+B(1:N))/2.0
      C (1:N)=T1(1:N)*T1(1:N)*3.14
```

(1)図1(1)の配列記述

```
      DO 10 I=1,N
       T1(I)=(A(I)+B(I))/2.0
   10 CONTINUE
      DO 11 I=1,N
       C(I) =T1(I)*T1(I)*3.14
   11 CONTINUE
```

(2)実行イメージ（スカラ）

```
      DO 10 I=1,N,VL *1
       T1(I:I+VL-1)=
     &          (A(I:I+VL-1)+B(I:I+VL-1))/2.0
   10 CONTINUE
      DO 11 I=1,N,VL *1
       C(I:I+VL-1)=
     &          T1(I:I+VL-1)*T1(I:I+VL-1)*3.14
   11 CONTINUE
```

(3)実行イメージ（ベクトル）

図2　配列記述とその意味

【注意 *1 】FACOM VPシリーズのようなベクトルレジスタ（vr）を持つ並列計算機において、ベクトルレジスタの最大長を超える回転数を持つループに対しては、ここで示されているような制御ループが必要である。

しかし、実際に一つの文を一つのDOループに展開して実行すると、ループの初期化や回転のオーバヘッドが大きいため、DOループ内に複数の文を記述した場合と比較して、性能は著しく低下することになる（図1、図2参照）。

ここで、従来の仕様で記述した複数の文から構成されるDOループに対する実行イメージ（図1の(1)(2)）と、配列記述から得られる実行イメージ（図2の(2)(3)）と

を比較すると、実行時間（ループが二つ）、オブジェクトサイズ（T1が単純変数から配列に変更）ともに配列記述の方が劣ることが明らかである。

配列記述を導入して記述性を高めても、そのために性能が低下しては科学技術計算向けのプログラミング言語としては、価値がない。従来のFORTRAN 以上の性能を出すために、コンパイラによりループを融合し、図1(2)のような命令列を生成する必要がある。

ループを融合すると、二つの効果が得られる。一つは、ループのオーバヘッドの解消であり、もう一つは、中間結果を保持する配列の削除である。中間結果を保持する配列が削除されると、ストア・ロードの時間が減るだけでなく、配列の領域が削減されるので、実行時間、オブジェクトサイズともに改善される。

しかし、ループの融合は、常に可能とは限らず、ループの回転数の一致（図３）、融合可能なデータ引用関係（図４）を持つことが必要である。データ引用関係における融合の可能性を判定するためには、ループ内で使用される全変数に関して、融合前と同じ定義・参照の引用順序関係が保たれることを確認する必要がある。

コンパイラがループ融合を行って初めて、従来のFORTRAN 77と同等の性能が得られる。しかし、そのためのコンパイラの負担は大きく、翻訳時間の増加を招くことは免れない。ループ融合のための条件判定は自動ベクトル化と同様であるので、ベクトルマシン向けの場合は従来のFORTRAN 77/VP と同程度で済むが、スカラマシン向けの場合には、もともと自動ベクトル化のような処理が不要だったので影響が大きい。

以下に、ループ融合の可能性判定の論理と、翻訳時間低減のための工夫について述べる。

```
      DO 10 I=1,100
       . . .
   10 CONTINUE
      DO 11 I=1,100,2
       . . .
   11 CONTINUE
```

図３　ループ融合不可の例１

```
      DO 10 I=1,10
        A(I) =I
          p
   10 CONTINUE
      DO 20 I=1,10
        B(I)=A(I+1)
             q
   20 CONTINUE
```

実行結果が異なる

```
      DO 10 I=1,10
        A(I) =I
          p'
        B(I)=A(I+1)
             q'
   10 CONTINUE
```

図４　ループ融合不可の例２

3.　ループ融合の可能性の判定

ループの融合を行う時に大切なのは、配列要素の引用順序である。引用順序の解析論理は自動ベクトル化と共通している。任意の配列要素に着目して、そのままのループの実行による配列要素の引用順序と、ループを融合した後の実行による引用順序が同じでなければいけない。

図４のDOループの融合の例で考えてみる。

たとえば、A(2)という要素に着目してみると、融合する前のプログラムでは、p で定義された後、q で参照されている。以後、このような引用順序の関係を、p →q と書く。

それに対して融合した後のプログラムでは、I=1 の時にq'で参照されI=2 の時にp'で定義されている。つまり、q'→p'という引用順序関係が存在することにな

り、融合する前と異なる。よって、この場合はループ融合ができないことを示している。

そこで、ループ融合する前に、ループの実行における引用順序を解析する必要がある。解析した結果の配列要素の引用順序関係を、単純化した形で表現したものを引用順序関数dep(p,q)と書くことにする。

もともとの引用順序関数とは、同一ループ内に出現する配列要素をアクセスする時間の関係を表すものであり、引用順序関数の値がｎのときは、ソース上で先行する使用からｎ回ループが回転した後に後続の使用が起こることを表す。

ループ融合は複数ループを一つにまとめる処理を行うので、複数ループにまたがっての引用順序関数の値を計算しなければならない。

ループ間の引用順序関数値とは、２つのループを融合したと仮定して求めた引用順序関数値のことであり、それが非負の値ならば先行する変数の使用が後続の変数の使用よりも必ず先に起こるので、融合が正しいことを示す。つまり、ループ融合では引用順序関数値の符号が分かれば十分であり、具体的な数値は必要でない。場合によっては符号は分かるが数値までは求められないという事もある。その場合は、特別な引用順序関数値‘＋’あるいは、‘－’というものを考える。‘＋’は引用順序関数の値の符号が正であることを表し、‘－’は負であることを表す。特別な場合として、二つの引用がそれぞれ別々の配列要素を引用し、一つも同じ要素を引用しない場合がある。このような時の引用順序関数の値はφと表し、引用順序関係を持たないことを意味する。

ソースプログラム中の二つの配列要素の引用で、ある要素では先行する引用の後に後続する使用が起こり、別の要素では後続する引用が起こってから先行する引用がなされる場合には、p →q およびq →p の両方の引用順序関係が存在することになる。このような場合、引用順序関数の値は‘？’で表し、引用順序が不定であることを意味する。

以下、ループ間にまたがっての引用順序関係の情報収集の方法を述べる。ただし、ここでは配列記述をループと同値とみなして使用する。

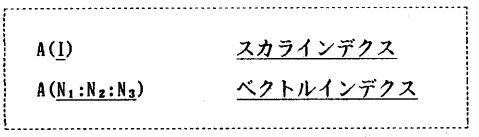


図５　インデクスの種類

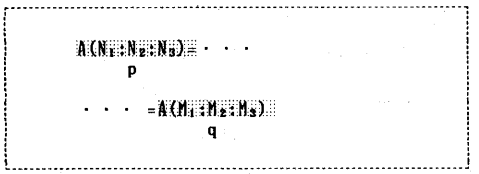


図6.　ベクトルインデクス同士の例

引用順序関数の値はいつでも求められるとは限らない。一方で定義された配列要素が、いつ、もう片方の引用で参照されるかということは、その時々の状態に依存し一般には分からないことがあるからである。求められない場合、結果を保証するためにdep(p,q)＝‘？’として取り扱わざるを得ない。以下では、値が求められる場合を例にして説明する。

まず、配列要素引用に現れるインデクスの種類（図５）によって、解析するインデクスの組み合わせは３種類に分けられる。

－　ベクトルインデクスとベクトルインデクス
－　スカラインデクスとベクトルインデクス
－　スカラインデクスとスカラインデクス

１）ベクトルインデクスとベクトルインデクス

この場合、二つのループが並列していると考えられる。引用p をDOループに展開した時のインデクスをi ，引用q を展開した時のインデクスをj とする。i とj は異なった範囲を動くので、融合した際の共通インデクスとして1,2,3,...,L の範囲を動くインデクスk を用いて考える。このインデクスが、融合した時のインデクスになる。i,j をk で表すと、

$i(k)=ak+b$

$j(k)=ck+d$

ただし $a=N_3$

$b=N_1-N_3$

$c=M_3$

$d=M_1-M_3$.

このように表しておくと、

$i(k)=j(k')$,$1\leqq k,k'\leqq L$

を満たす(k,k')のすべての組に対し、

$k'-k$＝定数 ならば $dep(p,q)=k'-k$

＞0 ならば ＝‘＋’

＜0 ならば ＝‘－’

である。その他の場合は$dep(p,q)$＝‘？’とする。

具体的には、表１に示すような条件の場合に$dep(p,q)$を求めることができる。

２）スカラインデクスとベクトルインデクス

この場合、引用順序関数の値が計算できるのは、図７のようにベクトルインデクスの初期値または終値とスカラインデクスの関係が明らかな場合である。

I の取り得る任意の値I_1における配列A での定義と参照の様子を図で表すと図８のようになる。

融合前はq の引用は１回であるが、融合後はp の要素数（回転数）だけ行われる。融合後のすべてのq の引用に先立ってその要素に対するp の定義が行われれば、$dep(p,q)$＝‘＋’であり、そうでなければ、$dep(p,q)$＝‘？’である。図７の場合は、$A(I_1)$ のpでの定義が$A(I_1:N_2:N_3)$ の先頭要素として行われてから、q でのA(I_1) の参照が行われるので$dep(p,q)$＝‘＋’となる。

３）スカラインデクスとスカラインデクス

スカラインデクスとスカラインデクスの解析は、図９のような場合に必要である。つまり、二つのループを融合する際に、二つのスカラインデクスを持つ配列への定義・参照が同一のループの中に取り込まれる場合である。

もともとは、１回の定義および参照であったのがループの回転数分だけ交互に定義・参照されるので、

表1. ベクトルインデクス同士の引用順序関数値

		dep(p,q)
a=c	(b-d)/aが整数	(b-d)/a
	(b-d)/aが非整数	ϕ
a≠c	a-c+b-d≧0かつ (a-c)L+b-d≧0	‘＋’
	a-c+b-d＜0かつ (a-c)L+b-d＜0	‘－’

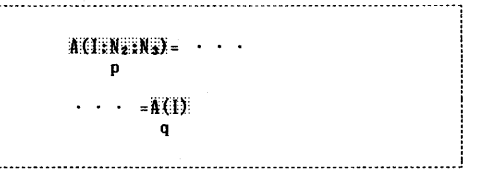


図７ スカラインデクスとベクトルインデクスの例

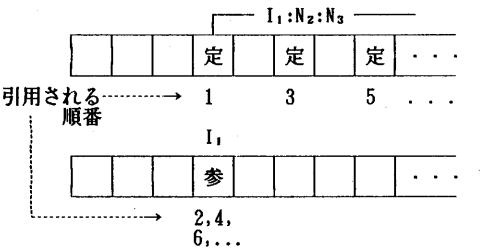


図８ 融合後の配列Ａの使用状況

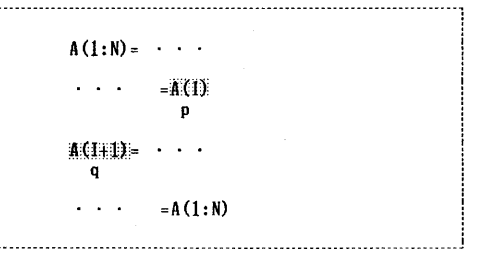


図９ スカラインデクスとスカラインデクスの例

dep(p,q)＝‘？’であり、pとqで同一の配列要素をアクセスする場合は融合不能である。pとqの添字の値が異なることが明らかな場合のみdep(p,q)＝‘φ’である。

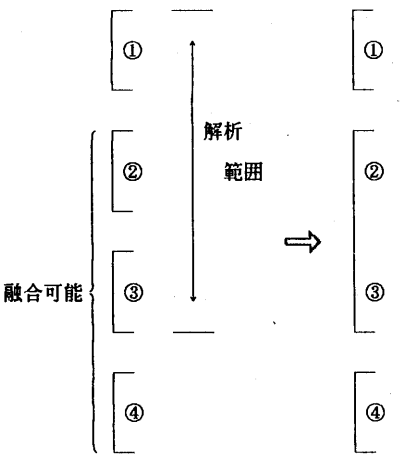
4. 引用順序関数計算の範囲

ループ融合のために引用順序関数を計算する場合、解析する範囲はおのずと限定される。回転数の違うループの場合はもちろん、ＧＯＴＯ文による飛び出しや飛び込みの前後で引用順序に関する情報を得ても、ループ融合は不可能なので意味が無く、その前後で範囲を区切ることになる。

しかし、そのように範囲を限定したとしても、ループ融合できる可能性のある範囲すべてにわたって解析することは、コンパイラにかなりの負担を負わせることになる。2.で述べた引用順序関数値を求めるためには、ループ融合をしようとする範囲にある各変数のすべての引用の組に対して、一つずつ添字式の解析を行う必要がある。ある変数に対する引用順序関数計算の対象の組み合わせの数は、融合の可能性を判定する範囲にその変数が出現する回数をｎとすると、 ${}_nC_2 = n(n-1)/2$であるから、ほぼ範囲の広さの二乗に比例する。実際のプログラムを見ても、解析範囲中のソースプログラムの量の二乗に比例して解析時間及び翻訳領域が必要になると予想される。つまり、ある程度以上広い範囲に対しては、処理時間がかかり過ぎ、実質的には融合処理ができなくなる。したがって、むやみに解析範囲を広げることはできず、ある程度の上限を設けてその中で解析することになる。

しかし、このように解析範囲をせばめると、本来融合できるはずだったループを、コンパイラは融合できなくなることがあり、実行性能を低下させることになる（図10参照）。

以上述べたように、ループ融合においては、翻訳資源（翻訳領域の大きさ・解析時間）と実行性能のトレードオフをどうするかが非常に大きな問題である。



②、③、④のループは一つのループに融合できるのだが、解析範囲を①、②、③に限定すると②、③のループしか融合されない.

図10 引用順序解析範囲と融合可能性

5. 最適化制御行（ＯＣＬ）による融合の指示

4.では、翻訳資源の問題からループ融合ができない場合があることを述べた。それ以外にも融合できないことがある。たとえば、引用順序関数の計算を行おうとしても、インデクスが変数の内容に依存するような場合には引用順序関数の値が求められず、融合処理は不可能である。

このように、引用順序関数の計算を自動的に行い、ループ融合を行うには、翻訳資源と論理の両面から、限界があり、その結果、融合処理自体にも限界がある。しかし、利用者からの要求は、できる限り融合を行って、実行時間を短縮し、オブジェクトサイズを小さくすることである。

この目的の為に、プログラマがコンパイラに情報を渡し、コンパイラの負担を小さくし、かつ高い性能を得ることができるようにする機能として最適化制御行が考えられる。

FORTRAN 77/VP で実現した最適化制御行は、ループ単位に指示を出し、ベクトル化のための情報をコンパイラに渡すためのものであった。それに対し、ループ融合で必要な最適化制御行は、ループ融合可能な文の範囲をコンパイラに示すものである。この最適化制御

行の導入により、引用順序関係の解析は一切不要になるので、翻訳時間が大幅に短縮される。

図11は、プログラマがソースプログラムを作成する際に、融合可能な範囲に対して、最適化制御行で予め融合可能であることを示している例である。*VOCL FUSIONから*VOCL ENDFUSION までが一つのループになることを示している。この例の融合可能な範囲には同一配列が数多く出現しているので、引用順序関数計算を行うと、翻訳時間が多くかかる。しかし、最適化制御行で融合が可能であることが示されているので、引用順序関数の計算を行わずに、高速に融合処理を行うことができる。

ただし、この指示を行った文が実際には融合不可能な引用順序関係を持っている場合、プログラムの実行結果は融合しない時と異なることになるので、注意が必要である。

```
*VOCL  FUSION
       AR(1:N)     =CX(5,1:N)
       BR(1:N)     =AR(1:N)-PX(5,1:N)
       PX(5,1:N)   =AR(1:N)
       CR(1:N)     =BR(1:N)-PX(6,1:N)
       PX(6,1:N)   =BR(1:N)
       AR(1:N)     =CR(1:N)-PX(7,1:N)
       PX(7,1:N)   =CR(1:N)
       BR(1:N)     =AR(1:N)-PX(8,1:N)
       PX(8,1:N)   =AR(1:N)
       CR(1:N)     =BR(1:N)-PX(9,1:N)
       PX(9,1:N)   =BR(1:N)
       AR(1:N)     =CR(1:N)-PX(10,1:N)
       PX(10,1:N)  =CR(1:N)
       BR(1:N)     =AR(1:N)-PX(11,1:N)
       PX(11,1:N)  =AR(1:N)
       CR(1:N)     =BR(1:N)-PX(12,1:N)
       PX(12,1:N)  =BR(1:N)
       PX(14,1:N)  =CR(1:N)-PX(13,1:N)
       PX(13,1:N)  =CR(1:N)
*VOCL  ENDFUSION
```

図11　最適化制御行の使用例
(Livermore loops No.10より)

6. まとめ

配列記述は、プログラムの記述性を高めるので、今後、科学技術計算用言語の主流となる仕様であると考えられる。現在、規格化が進んでいるFORTRAN 8Xにおいても、配列記述（FORTRAN 8Xでは配列区分という）が取り入れられている。当社でも、FORTRAN 77/VP ベクトル拡張言語に配列記述を導入し、FORTRAN 8Xの先取りを行ってきた。

しかし、高速性が要求される科学技術計算用に配列記述を使用するためには、配列記述仕様の持つ特性が、むしろ欠点になっている。この欠点を克服し、配列記述を真の意味での科学技術計算用言語に取り入れるには、コンパイラによる最適化が従来の言語仕様以上に必要であり、その結果、翻訳時間が増加する可能性がある。

本報告では、コンパイラによる自動ループ融合の論理を示すとともに、配列記述仕様向けの新しい最適化制御行を提案した。このような機能を持つ最適化制御行の必要性は、高速性が重視される言語にとってかなり高いものと考える。最適化制御行の導入により、配列記述仕様を持った言語に対する効率の良いコンパイラの実現が可能になるものと考える。しかし、最適化制御行は規格外であるため、プログラムの可搬性が損なわれるということがある。特にFORTRAN のように高速性が必須条件になっている言語においては、コンパイラやオブジェクトプログラムの効率を上げる機能（最適化制御行はその一例）が規格文法内に用意されるべきであると考える。

また、このような最適化制御行はプログラムの意味を変更してしまう可能性があり危険であるため、デバッグ機能等の考慮も重要な課題である。

今後、これまでの経験を活かし、スーパーコンピュータ向けの高速・高性能なコンパイラを開発していく所存である。

〔参考文献〕

(1)「FORTRAN 77/VP ベクトル拡張言語手引書」富士通
(2)平林，高嶋「ベクトルプロセッサとFORTRAN ベクトル拡張言語」情報処理学会プログラミング言語研究会 3-4,1985.12.13
(3)平林，赤松，高嶋「ベクトル記述を可能としたFORTRAN 拡張言語」情報処理学会第32回全国大会予稿集,1986,pp417
(4)L. Lamport,"The Parallel Execution of DO Loops",Comm.ACM 17,2(Feb.1974),pp83
(5)S.Kamiya,F.Isobe,H.Takashima and M.Takiuchi, "Practical Vectorization Techniques for the FACOM VP",Information Processing, ed. R.E.A. Mason, IFIP(1983) pp389
(6)堀田，磯辺，滝内，山梨「FACOM VPシステムにおける自動ベクトル化技術」昭和59年度電子通信学会総合全国大会予稿集（Ⅶ）,1984,pp233