スーパーコンピュータにおける コンパイラの最適化

岩沢 京子，田中 義一

1. はじめに

スーパーコンピュータと呼ばれる高速な演算処理能力を持つ計算機は，わが国ではベクトル計算機が主流となって商用化され実用に供されてきた．これらは，常にその時代の最高の計算パワーを必要とする大規模な科学技術計算の分野を中心に利用されている．蓄積されたソフトウェアの資産を生かすため，またこの分野のユーザーがもっとも慣れている点から，プログラミング言語としてFORTRANがサポートされてきた．

ベクトル計算機は，ベクトルデータと呼ぶ配列などのまとまったデータを，ベクトル命令によって実行させる．ベクトル命令は，ベクトル演算器によりパイプライン方式で高速に実行され，かつハードウェアにより内部的に並列に実行される［1］．自動ベクトル化コンパイラは，従来の逐次型FORTRAN言語で記述されたソースプログラムからベクトル処理可能な部分を抽出し，ベクトル処理を行なうようなベクトル命令を生成する［2］，［3］．

したがって，オブジェクトコードの実行性能を向上させるためには，ユーザーが記述したFORTRANソースプログラムからより多くのベクトル実行可能な部分を検出すること，これらから用意されたハードウェアの性能を十分に引き出すようなオブジェクトコードを生成することが重要となる．

本稿は，FORTRANコンパイラにおける最適化処理の1つである自動ベクトル化技術について紹介する．まず，**2.**で，ベクトル化可否の条件とデータ依存関係について，**3.**で，ベクトル化変換について，**4.**で，ベクトル最適化について紹介する．

いわさわ きょうこ，たなか よしかず
㈱日立製作所 中央研究所
〒185 国分寺市東恋ケ窪1—280

2. ベクトル化可否の条件とデータ依存関係

2.1 ベクトル化可否

ベクトル処理の対象は，実行比率が高い繰り返し計算で，原則として実行前に繰り返し回数が計算可能なDOループである．ベクトル処理では，ある程度まとまったデータを同時に，メモリからレジスタにロードし，演算を行ない，結果をメモリにストアする．したがって，スカラ命令により逐次に実行させる場合とでは実行順序が異なる．

図1-(1)にベクトル化可能な例を，**図1**-(2)にベクトル化できない例を示す．図1のDOループ中に2つの代入文（文1と文2）がある．スカラではループ制御変数Iの各値に対して文1と文2を順に実行するが，これをベクトル化すると，文1だけをループの全繰り返しについて実行し，次に文2をループの全繰り返しについて実行する．図1-(1)のように文2の代入左辺の配列Bの添字がI-1で，常に文1のB(I)の使用が定義に先立つ場合は，このままベクトル化しても定義使用の順序は変わらず，ベクトル化可能と判定する．図1-(2)の配列Bのアクセスのように，スカラ実行とベクトル実行で定義と使用の順序が異なる場合は，このままではベクトル化できないと判定しなければならない．このような判定は，配列の添字を解析して行なうが，一般化すると次のようになる．

ループ中の文1と文2に同じ配列へのアクセスがあり，スカラ実行する場合常に文1は文2に先行する．文1に現われる添字式1と文2に現われる添字式2がループの実行中に，重なることがあるか，重なる場合は文1のアクセスと文2のアクセスのいずれが先行するかを解析する．そのために，添字式1と添字式2をループ繰り返し回数の関数で標準化し，整数方程式「添字式1＝添字式2」がループ繰り返しとして意味のある範囲（1からループ長の間）で成立するかを調べる．成立しない場合は，配列名称は等しくてもアクセスする要素は異なる

DO10 I=1, N
A(I)=B(I)+C(I) ……文1
B(I−1)=D(I)+E(I)……文2
10 CONTINUE

(1) ベクトル化可能なループ

スカラの実行順序	ベクトルの実行順序
文1(I=1:B(1)使用)	文1(I=1:B(1)使用)
文2(I=1:B(0)定義)	文1(I=2:B(2)使用)
文1(I=2:B(2)使用)	・
文2(I=2:B(1)定義)	文1(I=N:B(N)使用)
・	文2(I=1:B(0)定義)
・	文2(I=2:B(1)定義)
文1(I=N:B(N)使用)	・
文2(I=N:B(N−1)定義)	文2(I=N:B(N−1)定義)

DO10 I=1, N
A(I)=B(I)+C(I) ……文1
B(I+1)=D(I)+E(I)……文2
10 CONTINUE

(2) ベクトル化できないループ

スカラの実行順序	ベクトルの実行順序
文1(I=1:B(1)使用)	文1(I=1:B(1)使用)
文2(I=1:B(2)定義)	文1(I=2:B(2)使用)
文1(I=2:B(2)使用)	・
文2(I=2:B(3)定義)	文1(I=N:B(N)使用)
・	文2(I=1:B(2)定義)
・	文2(I=2:B(3)定義)
文1(I=N:B(N)使用)	・
文2(I=N:B(N+1)定義)	文2(I=N:B(N+1)定義)

図 1 DOループにおけるスカラとベクトルの実行順序

ため，独立の計算であり，ベクトル化することができる．成立する場合は，解の大小から文1と文2のアクセスの順序を解析し，制御の流れと同様に文1が先行すればベクトル化可能と判定する．

同一ループ中に現われる同一配列名称に対する定義と使用，定義と定義の組について，上記のように各ループ繰り返しでアクセスする配列要素の重なりの有無，重なりがある場合にそのいずれのアクセスが先行するかを解析する [4]．

与えられたループのベクトル化可否を判定するのであれば，以上の解析で十分である．しかし，ベクトル化範囲を広げたり，一層性能向上を図るためのプログラム変換処理をほどこすためには，上記により解析した配列要素や変数のアクセス順序にいくつかの属性を付加したデータ依存関係を用いる．このデータ間の関係を有向線分で表わし，グラフ表現したものをデータ依存グラフという[5]．

2.2 データ依存関係

データ依存関係の種類には，図2-(1)に示す3種類がある．定義した値とその使用の順序関係を示すフロー依存，使用した値を定義によって更新する順序関係を示す逆依存，定義した値とその更新の順序関係を示す出力依存がある．さらに，これらの依存関係に図2-(2)に示すループ運搬依存の有無を解析して付加する．ループ運搬依存とは，ループの繰り返しが発生したときに生じる依存である．文1のA(I)の定義から文2のA(I)の使用へフロー依存があるが，これはループの繰り返しとは関係なく常に同じ回で定義した値を使用している．このような依存をループ独立な依存という．

一方，文1のA(I)の定義から文3のA(I−1)へのフロー依存は，ループI回目に定義した値を次のループI+1回目で使用している．このような依存をループ運搬依存という．お互いにループ運搬依存しかない2つの文は入れ換えても，定義や使用の順序が変わらないため，計算結果は等しい．

文4や文5は変数の例である．文4のSの定義から文5のSの使用へのフロー依存はループ独立である．変数の場合は，文5で使用したSを次のループ繰り返しの文4のSの定義で更新するからループ運搬逆依存も存在する．

このようにしてデータ依存グラフをつくるが，これをより正確なものにすることが，ベクトル化を含めた最適

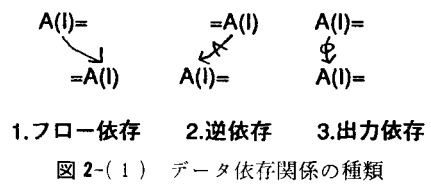


図 2-(1) データ依存関係の種類

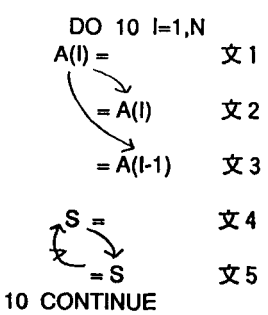


図 2-(2) ループ運搬依存の例

図 2 データ依存関係

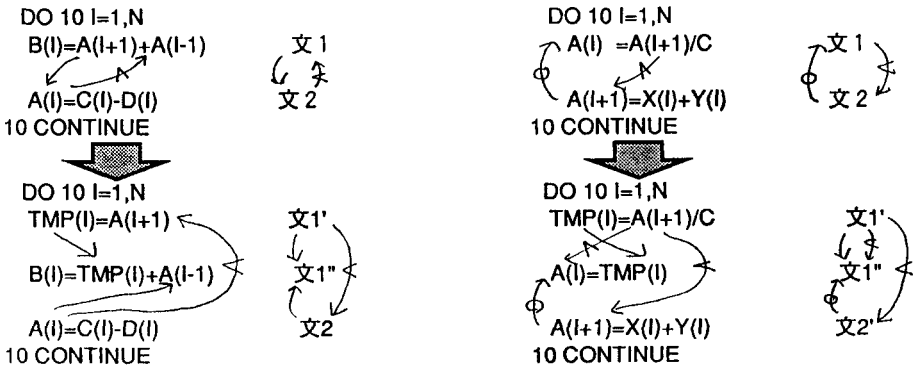


図 3-(1) 逆依存分割（左図）と出力依存（右図）の例

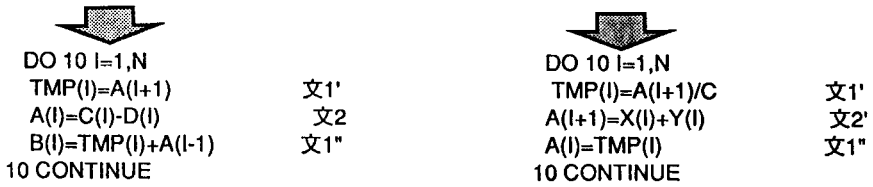


図 3-(2) ベクトル化のための文の並べ換え

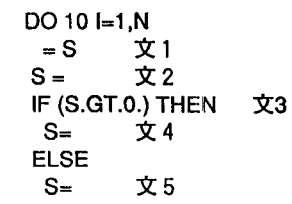
化処理の適用範囲を広げることにつながる．そのため，条件分岐などを考慮したり，ループ外の情報を取り入れるなど大域的なデータフロー解析が必要となる［6］，［7］．

3. ベクトル化変換

コンパイラは，前章で説明したデータ依存グラフを用いてベクトル化のためのプログラム変換をほどこす．以下のような手順でベクトル化できない部分を局所化させていく．

3.1 強連結成分の解消

データ依存グラフがサイクルを構成する強連結成分は本質的にベクトル処理を行なうことができない計算である．しかし，強連結成分を構成する有向線分の1つに逆依存や出力依存を含む場合は，一時的な配列に値を退避させることによりサイクルを壊すことができる．図 3-(1)，(2) に逆依存と出力依存を2つの依存に分割して強連結成分を解消する変換を示す．逆依存の場合は，始点の使用を一時変数に退避する代入文をつくり，使用をその一時変数で置き換える．出力依存の場合は終点の定義を一時配列に置き換え，この一時配列からもと定義への代入文をつくる．



このように強連結成分を解消すると，3.3 で述べる文の並べ換え処理によりベクトル化可能とすることができる．フロー依存のみから強連結成分が構成される場合はこのような解消を行なうことはできない．

3.2 変数の名称変換と配列化

図2に示したように，単純変数は複数回ループの中に現われると，データ依存グラフは必ず有向線分のサイクルを構成する強連結成分となる．逆依存は，使用が終わるまで定義によって値が壊されてはいけないことを示しているから，逆依存の終点の定義では，始点の使用とは異なる変数に代入すればよい．そのためコンパイラは図4に示すような変換を行なう．フロー依存でつながる変数ごと（図4では文1と文4と文5，文2と文3の2種）に変数を一時配列で置き換え，適宜添字を付加する．必要であれば，その一時的な配列にループ直前の変数の値

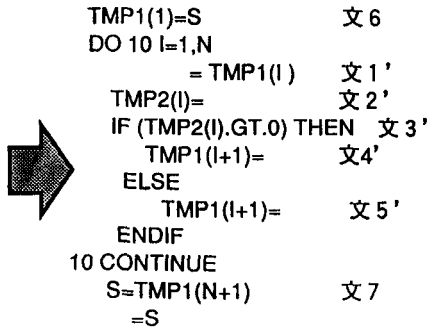


図 4 変数の名称変換と配列化

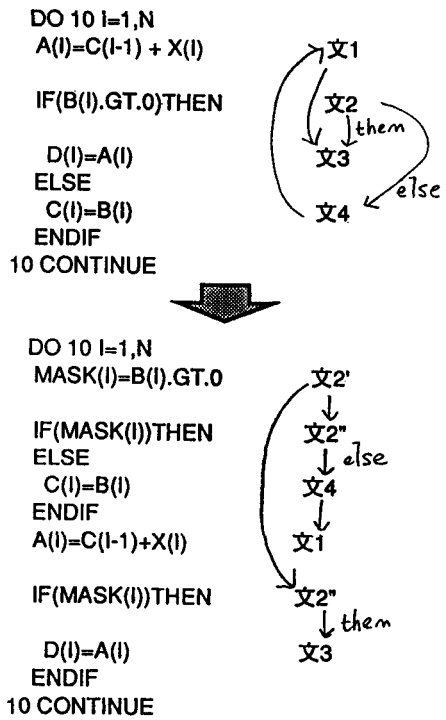


図 5 条件文付きの文の並べ換え

を代入する文（図4の文6）や，一時的な配列の最終要素を元の変数に代入する文（文7）をつくる．

このような変換により逆依存が解消されるため，データ依存グラフの強連結成分をなくすことができる．

3.3 文の並べ換え

2.に記したように，ループ繰り返しによる制御の流れと反対方向のループ運搬依存があるとベクトル化することはできない．しかし，ループ運搬依存のある文はそのループの中で入れ換えても，定義と使用の順序は変わらないため，ベクトル化可能となるように並べることができる．**3.2**で示した図3-(1)の例では図3-(2)のように並べ換えるとベクトル化可能となる．このような並べ換えは条件文があっても行なうことができる．

ベクトル処理においては，ＩＦ文はスカラ処理のように実際に条件を判定して分岐するのではなく，条件の成否を0と1で表わしたマスクベクトルと呼ぶデータをつくり，ＴＨＥＮ側，ＥＬＳＥ側とも，マスク命令と呼ばれるマスクベクトルが1の要素のみ（または0の要素のみ）処理する命令によって実行される．必要であれば条件分岐を複写して文の移動を行なう．このような場合は，**2.**に記したデータ依存関係だけでなく，条件文からその条件下の文への制御依存を導入して移動の可否の判定を行なう．**図5**に例を示す．文2′でマスクベクトルを作成すると，文4や文3はマスク付きの演算を行なう．

また，ベクトル化できない強連結成分を小さくするために，強連結成分の内側にあるが，直接強連結成分を構成しない文や，関係のない演算を外に追い出す移動を行なう．

このような文の並べ換えには，いくつかの自由度がある場合がある．そのときは，なるべくベクトル化可能な文がまとまるよう配置する．これは，次節で述べるベクトル化のためのループ分割の回数を最小限にするためである．ループ分割の目的は，1つのループにベクトル化可能な部分と不可能な部分が現われたときに，ベクトル化可能な部分を独立したループにして，ベクトル処理を行なうことである．しかし，ループを分割すると，新たに本来の計算には不要なオーバーヘッドを生むこと，また，ベクトル演算の開始には起動オーバーヘッドがかかるため，できるだけベクトル化可能な文をまとめることが性能を向上する上で重要となる．

3.4 ループ分割とループ交換

ループ分割は，1つのループを文と文の間で分けて複数のループにすることで，ループ全体ではベクトル化できないがその一部をベクトル化する場合や，後述するループ交換やループ一重化や外側ループ展開のために密多重化する場合，外側ループの最内部をベクトル化する場合などに必要な変換である．**図6**に外側ループを分割して密多重化した例を示す．

ループ分割可否は，分割点をまたがって後方から前方へのループ運搬依存の有無による．また，変数の配列化や，外側ループについて分割する場合は配列の次元を1つふやすことによって，後方から前方へのループに運搬される逆依存や出力依存があってもループ分割を行なうことができることもある．

ループ交換は，外側ループと内側ループを入れ換えることで，一般に密多重ループでなければ適用できない．ループ交換の十分条件の1つは，「データ依存グラフにおいて，外側ループに関するループ運搬依存がないこと」で，このとき最内側ループと外側ループは交換することができる［9］．したがって，最内側ループがループ運搬フロー依存によって強連結成分を構成してベクトル化できないが，外側ループにはループ運搬依存がない場合は，交換可能で，交換すると強連結成分は最内側ループにはないため，ベクトル化可能となる．図6にルー

```
   DO 10 I=1,N
     A(1,I)=X(I)*Z(I)-Q
     A(2,I)=Y(I)*Z(I)-P
     DO 10 J=3,M
       A(J,I)=(A(J-2,I)+A(J-1,I))/2.
10 CONTINUE
```

ループ分割

```
  DO 10 I=1,N
    A(1,I)=X(I)*Z(I)-Q
    A(2,I)=Y(I)*Z(I)-P
10 CONTINUE
  DO 10' I=1,N
    DO 10' J=3,M
      A(J,I)=(A(J-2,I)+A(J-1,I))/2.
10' CONTINUE
```

密多重ループ

ループ交換

```
  DO 10 I=1,N
    A(1,I)=X(I)*Z(I)-Q
    A(2,I)=Y(I)*Z(I)-P
10 CONTINUE
  DO 10' J=3,M
    DO 10' I=1,N
      A(J,I)=(A(J-2,I)+A(J-1,I))/2.
10' CONTINUE
```

内側ループのベクトル化可能

図 6　ループ分割とループ交換

プ分割した後のループ交換の例を示す．

また，ベクトル処理においては，ループ長が十分大きくなければハードウェアの性能を十分に引き出すことができない．そこで，内側ループも外側ループもベクトル化可能な場合は，ループ長の長いループを内側へ移動する交換を行なうこともある．

4.　ベクトル最適化

基本的には従来のスカラ計算機に対して行なってきた最適化 [10] は，ベクトル処理にとっても重要である．特に，配列要素の共通化は，ベクトルデータのロード・ストアによるメモリアクセスの回数を減らすことにつながり，性能向上に大きな影響を与える．

本章では，ベクトル固有の最適化項目をいくつか紹介する．ただし，ベクトルレジスタ割当など，機械依存の最適化も重要であるが，ベクトル計算機の構成や特性に依存するためここでは触れない．

4.1　ループ一重化

これは，多重ループを一重化することによりループ長を大きくしてベクトル実行の効率をあげる変換である．これはループの制御変数が変わるため，配列要素のアドレス計算も変える必要がある．このアドレス計算が新たなオーバーヘッドとならないよう，図 7-(1)のように全要素をアクセスすることを適用条件にしたり，図 7-(2)のように間接指標をデータ文のようにしてコンパイル時に生成し，これを用いて配列のアドレスを計算するような方法がとられている [11].

```
   REAL A(10,10) , B(10,10)
   DO 10  J=1,10
      DO 10 I=1,10
        A(I,J) = B(I,J)+X
10  CONTINUE
```

```
   REAL  A(10,10) , AA(100) , B(10,10) , B(100)
   EQUIVALENCE(A(1,1),AA(1)),(B(1,1),B(1))
   DO 10 K=1,100
     AA(K)=BB(K)+X
10  CONTINUE
```

図 7-(1)　配列の全領域アクセスする場合の一重化

```
   REAL A(10,10) , B(10,10)
   DO 10  J=1,3
      DO 10 I=1,I
        A(I,J) = B(I,B)+X
10  CONTINUE
```

```
   REAL  A(10,10) , B(10,10)
   DATA  II(6)/1,1,2,1,2,3,/
   DATA  JJ(6)/1,2,2,3,3,3,/
   DO 10 K=1,100
     A(II(K),JJ(K))=BB(II(K),JJ(K))+X
10  CONTINUE
```

図 7-(2)　間接指標による一重化

4.2　外側ループ展開

ループを展開するのは，複数のパイプラインが並列に動作し同時に実行されるベクトル演算が増えるよう，ループ内の演算量を増やすのが目的である．このとき，内側ループを展開すると，ループ内の演算は増えてもループ長が短くなるため十分な効果が期待できなくなる恐れがある．ベクトルループのループ長を減らすことなく演算を増やすために外側ループを展開する．

図 8 に例を示す．この例を見てもわかるとおり，最内側ループのループ制御変数が更新される前に，展開した外側ループのループ制御変数が更新されているため，展開する外側ループと最内側ループがループ交換可能でなければ，外側ループ展開は適用することができない．

4.3　ベクトル化の実行時判定

ベクトル処理にはそのための準備命令の実行やベクト

```
DO 10 J=1,M
DO 10 I=1,N
  A(I,J)=A(I,J)+X(I,J)*Y(I,J)
10 CONTINUE
```

```
DO 10 J=1,M,2
DO 10 I=1,N
  A(I,J)=A(I,J)+X(I,J)*Y(I,J)
  A(I,J+1)=A(I,J+1)+X(I,J+1)*Y(I,J+1)
10 CONTINUE
  IF (MOD(M,2).GT.0) THEN
    DO 11 I=1,N
      A(I,M)=A(I,M)+X(I,M)*Y(I,M)
11   CONTINUE
  ENDIF
```

図 8 外側ループ展開

```
SUBROUTINE SUB1(A,X,Y,N,M)
REAL A(N),X(N),Y(N)
DO 10 I=1,N
  A(I)=A(I+M)+X(I)*Y(I)
10 CONTINUE
END
```

```
SUBROUTINE SUB1(A,X,Y,N,M)
REAL A(N),X(N),Y(N)
IF((M.GE.0).AND.(N.GE.system-constant))THEN
  DO 10 のベクトル命令列
ELSE
  DO 10 のスカラ命令列
ENDIF
END
```

図 9 ベクトル化の実行時判定の例

ル演算器の起動処理などソースプログラムにはない処理が必要になる．このため，ベクトル化するループのループ長がある程度長くないとパイプライン処理の効果がでず，スカラの実行よりむしろ遅くなるケースがありうる．しかし，ループ長がコンパイル時に定数で与えられることは希であり，ループ制御変数でアクセスされる配列の宣言から推定できることもあるが，整合配列など推定の余地のないものもある．**2.** に記した配列の添字解析も同様で，コンパイル時に与えられた情報だけでは解析できず，そのためにベクトル化の可否が判定できないこともある．

このような場合，コンパイラは1つのループに対して，ベクトル命令列とスカラ命令列の両方を生成する．そして，ベクトル処理により実行効率が上がる条件や，データ依存関係がベクトル化可能となるような条件をつくり，実行時にこれらの条件判定を行ない，条件を満足する場合はベクトル処理を，満足しない場合はスカラ処理を行なうようなオブジェクトコードを生成する．このようにすると，常に条件判定が必要となるが，1つのループが異なるパラメタで何回か実行される場合でも，そのたびにスカラ・ベクトルの最適選択を行なえる．

図 9 にその例を示す．この例では，ループ長Nが，ベクトル処理がスカラ処理より速くなるシステム固有の定数値より大きいことと，ベクトル化するための条件として，配列Aの添字について常に$I+M \geqq I$が成り立つことである．

5. おわりに

スーパーコンピュータ向けのコンパイラの最適化技術として自動ベクトル化処理について紹介した．スーパーコンピュータの計算パワーを必要としたほとんどのアプリケーションが科学技術計算であったため，これらは，主に FORTRAN コンパイラの機能として発展してきた．

近年，メモリを共有したベクトル計算機を複数台並列に動かすスーパーコンピュータが相次いで発表され，コンパイラに自動並列化機能も提供されている．並列処理はオーバーヘッドが大きく，コンパイラは，より大きな粒度の並列処理単位を検出することが重要である．そのためには，ベクトル化のときの最内側ループ中心の解析ではなく，外側ループを中心とした広範囲な配列のデータ依存解析が必要である．また，並列実行比率を高めるための変換技術や，並列化のオーバーヘッドだけでなく，メモリアクセスの実行性能に与える影響も考慮した，変換技術が必要となる．

参考文献

[1] 村田健郎，小国　力，唐木幸比古著「スーパーコンピュータ」，丸善 (1985)，p. 304.

[2] Gotou S., Tanaka Y., Iwasawa K., Kanada Y., Aoyama A., : Advanced vectorization Techniques for Supercomputers, J. Inf. Process., Vol. 11, No. 1, pp. 23-31 (1987).

[3] 田中義一，岩沢京子：ベクトル計算機のためのコンパイル技術，情報処理，Vol. 31, No. 6, pp. 736-743 (1990).

[4] Takanuki R., Umetani Y., Nakata I., : Some Compilling Algorithm for an Array Processor, 3rd USA-Japan Computer Conference, pp. 273-279 (1987).

[5] Kuck D. J., Kuhn R. H., Padua D. H., Leasure B., Wolf M.: Dependence Graphs and Compiler Optimizations. Proc. 8th Annual ACM Symposium on Principles of Programming Languages. pp.177-189 (1981).
[6] 金田 泰，石田和久，布広永示：配列の大域データフロー解析法，情報処理学会論文誌，Vol.28, No.6, pp.567-576 (1987).
[7] 金田 泰，石田和久：到達定義，露出使用にもとづくデータ依存グラフの生成法，情報処理学会プログラミング言語研究会，87-PL-7 (1987).
[8] Tanaka Y., Iwasawa K., Umetani Y., Goto S.,: Compiling Technique for First-Order Liner Recurrence on a Vector Compiler, The Journal of Supercomputing, 4, 63-82 (1990).
[9] Allen J. R., Kennedy K.: Automatic Loop Interchange, Proc. of ACM SIGPLAN '84 Symposium on Compiler Construction, SIGPLAN Notices, Vol.19, No.6, pp.233-246 (1984).
[10] Aho A. V., Ullman J. D.: Principles of Compiler Design, Addison-Wesley (1977).
[11] Tsuda T., Kunieda Y.: Mechanical Vectorization of Multiply Nested DO Loops By Vector Indirect Addressing, Information Processing 86, North Holland, pp.785-790 (1986).

•ミニ•ミニ•　　•OR•

赤外線感知器

赤外線感知器をつけた装置は，今日，ごくありふれたものとしてわれわれの身のまわりにいくらでも見ることができる．テレビや冷房装置の遠隔操作，前に立てば，音もなく開く扉，手を出せば水の出る蛇口等々，何だかこちらの心の中まで見透かされているようで気味が悪くなるほどだ．しかし，こういう赤外線の利用は人手や資源を節約し，同時に秩序を維持する，人間の高度な知恵といえよう．

とはいえ，赤外線感知器の利用は平和的なものに限られるわけではない．草木で上手にカモフラージュされた戦車も，エンジンのぬくもりが出す赤外線で上空から探知されてしまう．ジェット戦闘機を攻撃するミサイルも赤外線感知装置を使ってこれを追尾する．攻撃されるジェット機の側でも，機首を太陽に向けたり，強烈な赤外線を出す“おとり”を使ってミサイルをそらそうとする．

しかし，こういう利用法が人間の邪智の生物ばかりかというと，そうでもない．コブラなど，ある種の邪悪な蛇は赤外線感知機能をもっていて，生きた獲物が発生する赤外線をたよりにこれを襲うのだという．

バングラデッシュの首府ダッカの街で仕事を待つコブラとりの話を新聞で読んだ．庭にコブラが出るとそんな1人が呼ばれる．コブラとりはポーチに腰をおろし，タバコを吸い，哲学者よろしく，しばしば瞑想に耽った後，おもむろに懐から手鏡を取り出し，庭の草むらのあちこちを照らしてコブラを見つける．そして，槍を手にして隼のように走りよってコブラの頭を一突きに貫いてしまうというのだ．

察するに，コブラは鏡に反射される熱帯の太陽の赤外線を鋭く感知して，思わず鎌首を持ち上げてしまうのであろう．また，コブラとりの方でも，永年の経験から，コブラのこのような習性を熟知しているのであろう．命がけの知恵くらべである．　　（からくり堂主人）