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†

安村通晃††　梅谷征雄††
堀越彌††

自動ベクトルコンパイラは，FORTRAN 等の DO ループに対して並列性を検出し，ベクトルプロセッサのためのオブジェクトを出力するコンパイラであるが，従来の自動ベクトルコンパイラでは，各 DO ループはすべてベクトル化されるか，されないかのいずれかであった．すなわち，ベクトル化不能の要因が一つでも DO ループ中に含まれると DO ループ全体がベクトル化不能となった．本論文では，DO ループを自動的に分割し，ループ中のベクトル化可能部分を部分的にベクトル化する方式を検討し，その実現方法について述べる．ループ分割の自動化のためには，分割の細かさの選択と，プログラム変換の正しさの保証との二つの課題を解決しなければならない．前者については，文を単位とする分割でほぼ良好な結果が得られた．後者については，ベクトル化判定で使われるデータ参照解析を準用し，また，変数の配列化等の考慮を払った．この種のプログラム変換機能は，自動ベクトルコンパイラにおいて，今後重要性が増すであろう．

1. はじめに

近年，わが国内外で各種のベクトルプロセッサが広く使われ始めている[1),2)]．ベクトルプロセッサのコンパイラ，すなわちベクトルコンパイラの方式としては，ライブラリ方式，言語仕様拡張方式，自動ベクトル化方式などがある．

従来プログラムからの移行性* という観点でこの3方式を比較すると，ライブラリ方式や言語仕様拡張方式では，ベクトルプロセッサ向きのプログラムの書換えが必要であるのに対し，自動ベクトル化方式ではこの書換えは一般には不要である，という違いがある．しかし，従来の自動ベクトル化方式のコンパイラ（以下，自動ベクトルコンパイラと略す）では，ベクトル化対象の DO ループに一つでもベクトル化不能の要因があると，DO ループ全体がベクトル化されない，という欠陥があった．

ここでは，DO ループ中のベクトル化不能部分と可能部分の分割を自動的に行って部分ベクトル化する方式を考察し，自動ベクトルコンパイラの一部として実現する方法について述べる．

本論文では，まず初めに，自動ベクトルコンパイラの基本的機能である，ベクトル化の判定のためのデータ参照解析について述べ，次に，ループ分割の基本的要請から自動ループ分割の基本方針を示し，さらに，この基本方針に従ったループ分割の実現方法を述べる．最後に，本方式の評価を述べ，締めくくりとする．

2. 自動ベクトルコンパイラ

自動ベクトルコンパイラは，FORTRAN 等の DO ループをベクトル命令列に翻訳するが，その中心的な機能は，DO ループ中のデータフローを解析して，並列性を検出することにある[3)]．すなわち，従来スカラ命令での DO ループは，DO の各インデックス値に対して，DO ループ中の各文を一通り実行して次のインデックスへと繰り返すが，ベクトル命令においては，DO ループ中の各文（各命令）をまずすべてのインデックス値について実行した後に，次の文（命令）の実行へ移る（**図1**）．これをベクトル化による実行順序の変更とよぶ．

このベクトル化による実行順序の変更に対して，計算結果が変わらないことを，データ参照関係が適切である，という．逆に，データ参照関係が適切でないことを，データ参照関係不適である，という．**図2**に，データ参照関係不適の例を示す．

データ参照関係を，変数と配列に分けて述べるならば，データ参照関係不適であるとは，

（1） 同一変数の出現の中に，定義が含まれ，その最初の出現が定義でないとき，

あるいは，

（2） 同一配列名の異なる二つの出現に対して，少なくともその一方の出現が定義であり，プログラム上で実行順序が先行する配列要素の添字が，後

---

† Partial Vectorization Method for Automatic Vector Compilers by MICHIAKI YASUMURA, YUKIO UMETANI and HISASHI HORIKOSHI (Central Research Lab., Hitachi Co. Ltd.).

†† (株)日立製作所中央研究所

* portability

```
DO 10 I=1,N                          A(i)←B(i)*C(i) for i=1…N
A(I)=B(I)*C(I)        ⇨              D(i)←E(i)+F(i) for i=1…N
D(I)=E(I)+F(I)
10 CONTINUE          ベクトル化
```

図 1 ベクトル化による実行順序の変更
Fig. 1 Execution reorder for vectorization.

```
DO 10 I=1,N                          DO 10 I=1,N
A(I)=S*B(I)                          A(I-1)=B(I)*C(I)
S=C(I)+D(I)                          D(I)=A(I)+E(I)
10 CONTINUE                          10 CONTINUE
```

図 2 データ参照関係不適の例
Fig. 2 An example of invalid data reference relation.

```
DO 10 I=1,N                          DO 10 I=1,N
A(I)=FLOAT(I)                        A(I)=FLOAT(I)
C(I)=A(I)*B(I)        ⇨              10 CONTINUE
10 CONTINUE          自動分割        DO 11 I=1,N
                                     C(I)=A(I)*B(I)
                                     11 CONTINUE
```

図 3 ループ分割による部分ベクトル化の例
Fig. 3 An example of partial vectorization by loop splitting.

続する配列要素の添字よりも“小さい”とき，であるといえる．

より厳密なデータ参照関係の定義を付録Aに示す．なお，データ参照関係不適の場合でも，変数ではベクトル和や内積の場合は，また配列では Iteration* の場合は，それぞれ特殊な演算とみなし，ベクトル化可能とすることもある．また，条件文を含む DO ループでは，上述のデータ参照関係の定義を多少拡張する必要がある．

3. 自動ループ分割

自動ベクトルコンパイラでは，プログラムの特別な書換えなしにベクトル化することを原則とするが，従来は DO ループ中に一つでもベクトル化不能要因があると DO ループ全体がベクトル化不能となった．このため，他のベクトル化可能部分が含まれている場合にはプログラマが人手により分割するときもあった．しかし，この方法では，分割時にエラーが混入する危険性があり，しかも分割に手間がかかる．そこで，自動ベクトルコンパイラにおいては，DO ループ中のベクトル化可能部分を不能部分より抜き出して，自動的に部分ベクトル化する方式が望まれる．**図3**に，ループ分割による部分ベクトル化の例を示す．この例では，組込み関数 FLOAT がベクトル化不能であることを仮定している*．

図3の分割前のプログラムでは，DO ループ全体がベクトル化不能となったが，分割後のプログラムでは，前半の DO ループはベクトル化不能であるが，後半の DO ループはベクトル化可能となる．

ところで，すべての DO ループがループ分割の対象となるわけではない．**表1**に全面的ベクトル化不能の DO ループの例を示す．これらの全面的ベクトル化不能 DO ループ以外の最内側 DO ループがループ分割の対象 DO ループであるが，その対象 DO ループに対してベクトル化不能要因となるものを**表2**に示す．表2に示す項目をループ分割によりベクトル化不能部分として部分化する．なお，表1や表2に示すベクトル化不能要因は一つの例であり，ベクトルプロセッサの命令レパートリやベクトルコンパイラの技術水準・サポート範囲等で変わりうるものである．

表 1 全面的ベクトル化不能 DO ループの例
Table 1 An example of totally unvectorizable DO loops.

(1) ループ外への飛び出しを含むループ
(2) 副プログラム呼び出しを含むループ

表 2 部分ベクトル化不能要因の例
Table 2 An example of partially unvectorizable factors.

(1) 内部小ループ，逆方向分岐
(2) データ参照関係不適
(3) 未サポートの文（入出力文等）
(4) 未サポートのデータ型（文字型等）
(5) 未サポートの組込み関数，等

4. ループ分割の基本方針

ループ分割に対する基本方針を設定する上で考慮すべき点は，次の3項目である：

(1) ベクトル化可能部分を最大限拡げる．
(2) 分割によるオーバヘッドを最小限にする．
(3) 分割によるプログラム変換で結果が変わらない．

まず，(1)はループ分割の本来の目的である，ベクトル化可能部分を最大限拡げ，ベクトル化による性能向上を図ることである．次に(2)については，ベクトル化されずに残る部分に対しては，スカラ命令としてのループ判定部のオーバヘッドの存在が問題であり，したがってベクトル化不能部分が細分化されて数多く

* Iteration の定義は付録A参照．

* 不能要因は別の要因であってもかまわない．

```
      DO 10 I=1,N                DO 10 I=1,N
      S=FLOAT(I)                 S~(I)=FLOAT(I)
      A(I)=B(I)*S          ⇨     10 CONTINUE
   10 CONTINUE                     DO 11 I=1,N
                                   A(I)=B(I)*S~(I)
                                11 CONTINUE
                                   S=S~(N)
```

図 4 ループ分割に伴う変数の配列化
Fig. 4 The promotion of a variable to an array for loop splitting.

```
      DO 10 I=1,N                DO 10 I=1,N
      IF(J.EQ.M) GOTO 100        IF(J.EQ.M) GOTO 110
      A(I)=FLOAT(I)              A(I)=FLOAT(I)
  100 C(I)=A(I)*B(I)       ⇨  110 CONTINUE
   10 CONTINUE                  10 CONTINUE
                                   DO 11 I=1,N
                               100 C(I)=A(I)*B(I)
                                11 CONTINUE
```

図 5 分割点への分岐文番号の変更
Fig. 5 Alteration of the statement number to the split point.

残ることは性能上望ましくない．ところで一般的には，ベクトル化可能部分を拡げるために分割を細かくしすぎると，(2)の要請に相反してオーバヘッドが増え，逆に分割を粗くしすぎると，(1)の要請が成り立たなくなる．そこで(1)と(2)の兼ね合いを考えた，適正な分割の単位 (size of grain) を考える必要がある．ここでは，分割の単位は文単位とし，文の中での分割は考えないことにする．文単位では，できるだけ細かくベクトル化可能部分を抽出することにする．また，条件文の中での分割も行わないことにする．これは，(3)の要請と関連して，分割した場合条件式を繰り返すが一般には条件式中の変数・配列がその条件制御下の文中で変更される可能性があるからである．その他，分割に際しては，不必要な分割は行わないことにする．したがって，ベクトル化可能部分どうし，または，不能部分どうしは連続化させ途中で分割はしないことにする．

次に，(3)のプログラム変換の正しさ保証のために，まずループ分割により，分割点を境に文の実行順序が変わることから，2章で述べたベクトル化のためのデータ参照関係の検査と同様の検査を行う必要がある．すなわち，ループ分割の分割点をまたがる，同一変数間または同一配列間のデータ参照関係は適切でなければならない．ただし，ベクトル化不能部分内でのデータ参照関係は適切でなくともよい．

さらに，変換の正しさのためには，ループ分割点をまたがって定義・参照される変数は配列化し，各インデックス値に対応した変数の値を分割点をまたがって受け渡しする必要がある（**図4**）．

最後に，ちょうど分割点への前方分岐* は，**図5**に示すように，分岐点の文番号を変更しなければならない．

以上の考察の結果をまとめると，ループ分割の基本方針は以下のとおりとなる：

(1) 表1に示す DO ループを除く最内側 DO ループをループ分割の候補とし，表2に示すベクトル化不能要因を検出し，不能部分を部分化する．

(2) 条件文内での分割は行わない．条件文以外では文単位に分割する．

(3) 分割点をまたがって定義・参照される変数は配列化する．

(4) ループ分割間でのデータ参照関係は，これを保証する．

(5) 分割点への前方分岐は，分岐先を変更する．

(6) ベクトル化可能部分どうし，不能部分どうしはおのおの統合して連続しないようにする．

5. ループ分割の方式

前章で述べた，ループ分割の基本方針にもとづき，次の手順でループ分割の処理を行う．

(1) ベクトル化不能部分はできるだけまとめ，ベクトル化可能部分はできるだけ細分化して，分割の候補点を定める〔分割候補点の設定〕．

(2) データ参照関係の検査を行いつつ，分割部分（以下 S-BLOCK とよぶ）の整理統合を行う〔分割候補点の調整〕．

(3) 定義と参照が分割点をまたがる変数を配列化する〔変数の配列化〕．

(4) その他，分割点の決定，分岐先の変更，中間テキスト，テーブル類の変更〔分割オブジェクトの生成〕等．

以下，自動ベクトルコンパイラでの各手順の実現方式の詳細を節に分けて述べる．

5.1 分割候補点の設定

分割候補点の設定に先立って，まず制御構造の解析を行う．

DO ループ内の制御の流れは，点（ブロック）と有向辺とから成る有向グラフ（プログラムフロー）として表される．点とは，制御移動を内部に含まない，一つ以上の文の集りである．**図6**に有向グラフの例を示す．ベクトル処理のための有向グラフは，最適化に用

* forward jump

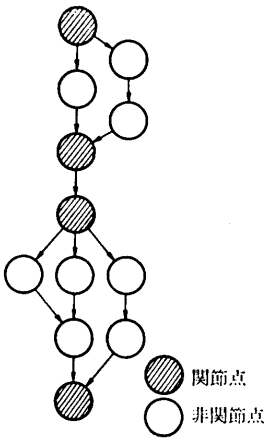

図 6 有向グラフの一例
Fig. 6 An example of the directed graph.

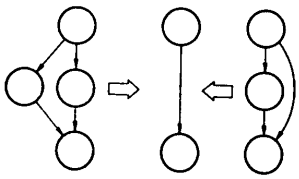

図 7 if-then-else 簡約
Fig. 7 If-then-else reduction.

いられるものとほぼ同一であるが，細かな点では違っている*.

この有向グラフに対して，始点と終点の間のすべての経路 (path) に対して，必ず通る点のことを関節点 (articulation point) とよび，そうでない点を非関節点とよぶ．図では関節点を黒丸で，非関節点を白丸で示した．

有向グラフ作成の後，関節点・非関節点の判別とサイクル（内部小ループ）の検出を行う．これらの制御構造の解析[4]は，有向グラフを深さ優先になぞる方式 (depth-first-traverse) を基本とし，これに if-then-else 簡約（**図 7**）による高速化技法を併用した．

この有向グラフと関節点の情報により，非関節点とその制御元である関節点（すなわち条件文の全体）を一つの分割の単位とし，他の一般の関節点（非条件文）はその構成要素である文単位に分割の単位とする．この分割の単位が S-BLOCK である．S-BLOCK に対して，表 2 に示すベクトル化不能要因に従って，ベクトル化の可否の情報を印づけする．その方法は，

（1） 内部小ループを構成する S-BLOCK はベクトル化不能にする．

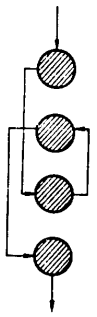

図 8 内部小ループでない逆方向分岐の例
Fig. 8 An example of cycle-free backward jump.

（2） 内部小ループでない逆方向分岐を構成する S-BLOCK もベクトル化不能とする（**図 8**）．これは，制御の流れとプログラム・テキストの順序が不一致となるためにループ分割を禁止する必要があるためである．

（3） 未サポートの組込み関数・データ型・文を含む S-BLOCK はベクトル化不能とする．

（4） ベクトル化不能の一連の S-BLOCK はまとめて，単一の S-BLOCK とする．

こうして，分割候補点が，S-BLOCK の形で設定された．この時点では，ベクトル化可能 S-BLOCK は，他のベクトル化可能 S-BLOCK と連続することがありうるが，ベクトル化不能 S-BLOCK どうしは連続していない．

5.2 分割候補点の調整

分割候補点の調整は，データ参照関係の検査と並行させて行い，その結果として S-BLOCK の整理統合を行う．データ参照関係の検査は，(1)変数，(2)配列，(3)特殊演算の順で行う．

（1） 変数のデータ参照関係の検査は，DO ループ全体にわたって行い，定義が含まれていれば先頭にあるかどうかを調べる．条件文中に最初の定義がある場合は，この検査を拡張する必要がある．データ参照関係が正しく，かつ，異なる S-BLOCK にまたがって定義・参照のある変数は，“配列化の可能性あり”との印づけを行っておく．データ参照関係違反の変数に対しては，その変数が最初に出現する S-BLOCK から，最後に出現する S-BLOCK まで，その前後のベクトル化不能 S-BLOCK(もしあれば) とともに，単一のベクトル化不能 S-BLOCK とする．

（2） 配列についてのデータ参照関係の検査は，同

* 最適化のための有向グラフと比べて，ベクトルコンパイラの有向グラフでは，たんに制御移動だけではなく，文を点（ブロック）の基本とする点で違いがある．

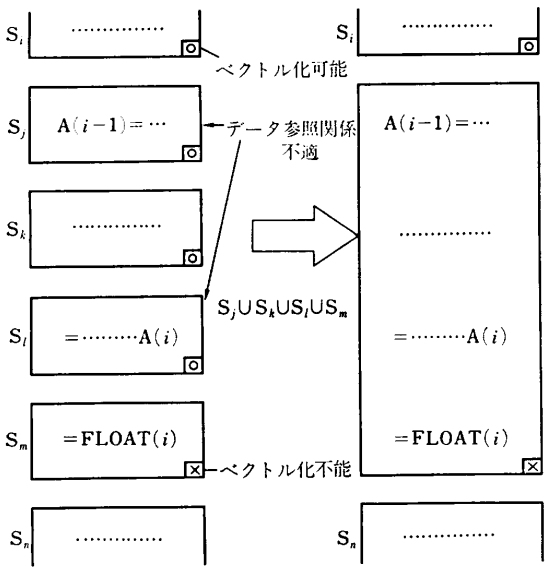

図 9 データ参照関係の検査と S-Block と統合の例
Fig. 9 An example of data reference check and S-Block integration.

一ベクトル化不能 S-BLOCK 内の二つの出現関係を除き，どちらかが定義であるような二つの出現の間の，添字の“大小”比較*を行い，データ参照関係の適否を判定する．データ参照関係不適となった S-BLOCK は，その途中の S-BLOCK と前後のベクトル化不能 S-BLOCK(もしあれば) とともに単一のベクトル化不能 S-BLOCK とする．

(3) 特殊演算のデータ参照関係は，ベクトル化可能な同一文の (したがって同一 S-BLOCK) 内のデータ参照関係を調べ，データ参照関係不適と判定された場合は，その前後のベクトル化不能 S-BLOCK(もしあれば) とともに，単一のベクトル化不能 S-BLOCK とする．

図 9 に，データ参照関係の検査と，その結果である S-BLOCK の統合の例を示す．図において S_i, S_j, S_k, S_l がベクトル化可能で S_m がベクトル化不能である時点で配列 A のデータ参照関係が調べられているとする．S_j における配列要素 $A(i-1)$ の定義出現が，S_l における配列要素 $A(i)$ の参照出現とデータ参照関係不適であるから，S_j と S_l を統合する必要がある．このとき S_j と S_l の間にある S_k，およびこれらと隣接するベクトル化不能 S-BLOCK S_m も統合する．結局，S_j と S_l 内相互のデータ参照関係不適の検出が S_j〜S_m までの S-BLOCK の統合をひきおこし，ベクトル化不能 S-BLOCK をまとめたことになる．

こうして，ベクトル化可能 S-BLOCK の内部，および異なる S-BLOCK 間で，データ参照関係の検査が終った後に，ベクトル化とループ分割の両方必要なデータ参照関係が保証されたことになる．この時点では，分割候補点の設定直後と同様，ベクトル化不能 S-BLOCK は連続していないか，ベクトル化可能 S-BLOCK は連続していることがありうる．

データ参照関係の検査の終了後は，

(4) ベクトル化可能 S-BLOCK で連続するものを統合する．

(5) CONTINUE 文のみ，などの演算を含まない，無効なベクトル化可能 S-BLOCK は除去する．

こうして，整理統合された S-BLOCK が，DO ループ全体で単一のベクトル化可能，または，ベクトル化不能の S-BLOCK となるとき，ループ分割は行わない．

ベクトル化可能 S-BLOCK とベクトル化不能 S-BLOCK がともに残ったときは，これらの S-BLOCK の境界をループ分割点として確定する．

5.3 変数の配列化

分割候補点の調整の際，変数の定義・参照が分割点をまたがるときは，配列化の可能性があるとして，その変数に対して印づけがなされた．分割点が確定したとき，単一 S-BLOCK に出現が限られる変数に対しては，この印づけを取り消す．こうして，ループ分割点が確定したとき，配列化変数も確定する．確定した配列化変数に対しては，配列化のための次の処理を行う：

(1) 新しい配列のための辞書テーブルの新設*.
(2) 配列化変数が出現する中間テキストの，新しい配列を用いた中間テキストへの変換.
(3) (必要に応じ)新しい配列用のインデックス変数の新設とそのインデックス変数の初期化中間テキスト，更新中間テキストの挿入.
(4) DO ループの直後に，配列化変数の終値保証用の中間テキストを挿入.

5.4 分割オブジェクトの生成

確定したループ分割点のおのおのに対して，中間テキスト，文番号，制御移動情報，等の更新を行う．そのおもな処理は以下のとおりである．

* 添字の“大小”比較は，添字を初期値表現，増分値表現，終値表現に分けて比較を行う．この比較の結果“大小”関係が不明となる場合については，データ参照関係が不適と同じ扱いをする．

* 配列のサイズは，ループ長，または，ループ内に現れる配列の宣言から定める．そのいずれもが不明の場合は，ユーザオプションの指定により決める．

(1) 分割直後への分岐文番号の変更

図5に例示したとおり，分割点直後への前方分岐を含む場合は，新しい文番号を分割点の直前に作り，分割点への分岐は，その新文番号への分岐となるように，中間テキストと制御情報の書換えを行う．逆方向分岐はそのままとする．

(2) 分割点での文番号辞書の新設

分割点が一連の代入文で文番号がない場合には，分割点の直後に，新しい文番号辞書を作り，制御移動情報を変更する．

(3) インデックス更新中間テキストの挿入

各 S-BLOCK の末尾(または，インデックス変数の型によっては先頭) に，その S-BLOCK 内で使われているすべてのインデックス変数の更新中間テキストを挿入する．

(4) 分岐中間テキストの生成

各 S-BLOCK の末尾に，その S-BLOCK の先頭文番号への分岐中間テキストを作る．その際，ループ制御変数の更新中間テキストも加える．また，制御移動情報の書換えも行う．

(5) インデックス初期化中間テキストの挿入

各分割点の直前に，その S-BLOCK 内で使われているすべてのインデックス変数の初期化中間テキストを挿入する．

(6) インデックス変数の終値保証中間テキスト挿入

ループ全体の直後に，新設インデックス変数の終値保証中間テキストを挿入する．

以上の処理において，ループの先頭 S-BLOCK と末尾の S-BLOCK とは，中間の S-BLOCK と多少異なる扱いとなる*.

6. 本方式の評価と今後の課題

本方式の自動ループ分割機能を HITAC M200H, M280H 等の IAP 用コンパイラの拡張機能の一部として実現した．その本格的な評価は今後の課題であるが，科学技術計算プログラム45題に対して適用状況の解析を行った．

45題ジョブ中ループ分割の対象になった DO ループが43個あり，さらにそのうちの23個が配列化変数を含んでいた．このことから，配列化変数は比較的出現頻度が高いことがわかる．また，これら43のループのうち，わずかに2ループのみが2分割で，残りすべてが1分割であることから，当初懸念していた細分化によるオーバヘッドの増大は大きな比重を占めないことがわかった．さらに，これらのループ分割適用 DO ループの分割後のベクトル化可能部分の比は，全体の平均が47%で約半分に近い値となった．

ループ分割による性能向上は，ループ分割を他の機能拡張とともに行ったために現時点では評価できない．これらは今後の課題である．

7. おわりに

文献5) は，いくつかのソースレベルでの変換によるプログラムの改良例を挙げている．しかしそこでも述べられているとおり，それらのうち，どの変換機能が有用であるかを示すのは容易ではない．とくに自動ベクトルコンパイラにおいては，プログラム変換に対して，常に変換の正しさを保証し，オーバヘッドに対してもガードを行う必要がある．その点，プリプロセッサ方式よりも厳しい条件が課せられている，といってよい．

ループ分割は自動ベクトルコンパイラの基礎技術として重要であり，今後このようなプログラム変換機能に対する要求はますます強まっていくと思われる．したがって，有用と判断されるプログラム変換機能は，自動ベクトルコンパイラの一部としてとり込んでいく必要があろう．

謝辞 本研究に関して，有益な討論や助言をいただいた，日立製作所ソフトウェア工場高貫隆司，小林里昭の両氏に感謝したい．

参考文献

1) Kozdrowicki, E. W. and Theis, D. J.: Second Generation of Vector Supercomputers, *IEEE Comput.*, Vol. 13, No. 11, pp. 71-83 (Nov. 1980).
2) 田中善一郎：科学技術計算用コンピュータにおける高速化手法，日経エレクトロニクス (Aug. 3, 1981)
3) Takanuki, T., Umetani, Y. and Nakata, I.: Some Compiling Algorithms for an Array Processor, Proc. of 3rd USA-JAPAN Comput. Conf., pp. 273-279 (1978).
4) 安村通晃，松永 通，田中義一，梅谷征雄：ベクトルコンパイラにおける制御構造解析の一方式，情報処理学会第23回 (昭和56年後期) 全国大会予稿集，pp. 211-212 (Oct. 1981).
5) Loveman, D. B.: Program Improvement by

* たとえば，末尾の S-BLOCK は，分岐中間テキストの挿入が不要である，など．

Source-to-Source Transformation, *J. ACM*, Vol. 24, No. 1, pp. 121-145 (Jan. 1979).
6) 梅谷征雄，高貫隆司，堀越　彌：ベクトルマシンとベクトル化コンパイラの方式，電子通信学会技術研究報告，EC 80-15, pp. 49-56 (Jun. 1980).
7) 梅谷征雄，高貫隆司，安村通晃：技術計算プログラムの自動ベクトル化技術，情報処理，Vol. 23, No. 1, pp. 29-40 (Jan. 1982).

付録A　データ参照関係の定義

（1） 変数Sがデータ参照関係不適であるとは，変数Sの DO ループ内におけるすべての出現 $S_k(k=1, \cdots, m)$に対して，

$$\neg\,\mathrm{DEF}(S_1)\ \&^{\exists} j(j \neq 1)\ \mathrm{DEF}(S_j)$$

であることをいう．

ただし，

$$\mathrm{DEF}(x)\begin{cases}=\text{true if} & x\text{が代入文の左辺に出現}\\ =\text{false otherwise}\end{cases}$$

（2） 配列Aがデータ参照関係不適であるとは，配列Aの DO ループ内におけるすべての出現 $A_k(F_k)$ $(k=1, \cdots, m)$に対して

$$(\mathrm{DEF}(A_i(F_i)) \vee \mathrm{DEF}(A_j(F_j))) \wedge ({}^{\exists} i, j\ F_i \lessdot F_j)$$

が成り立つことである．

ただし，$F_i, F_j (i<j)$は添字値の順序付き集合で，

$F_i=(f_i{}^1, \cdots, f_i{}^n)$

$F_j=(f_j{}^1, \cdots, f_j{}^n)$

nはループ回数，

$f_i{}^l, f_j{}^l$ は l 回目，添字値 $(1 \leq l \leq n)$

$$F_i \lessdot F_j \underset{\text{def}}{\Longleftrightarrow} {}^{\exists} p, q (1 \leq p < q \leq n)\ f_i{}^p = f_j{}^q$$

（3） データ参照関係不適の例外となる特殊演算 (reduction function) としては，たとえば，次の種類がある：

① ベクトル和(差)　$S=S \pm A_i$

② 内積(補の内積)　$S=S \pm A_i * B_i$

③ 1階の Iteration　$A_{i+1}=A_i * B_i + C_i$

(昭和57年2月26日受付)
(昭和57年6月15日採録)