Vol. 31 No. 6　　情　報　処　理　　June 1990

解　説

ベクトル計算機のためのコンパイル技術†

田中義一††　岩澤京子††

1. はじめに

スーパコンピュータと呼ばれる高速計算機のなかで，汎用目的のために実用化されているのがベクトル計算機である．この計算機は，従来の計算機と異なり，ベクトル命令と呼ばれる高水準の命令をハードウェアにより内部的に並列実行することにより，高性能化を実現している[1),2)]．このため，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る．しかし，ユーザがプログラムを書く際にアセンブラで直接ベクトル命令を用いることは以下の問題がある．つまり，プログラミングの生産性が低いこと，蓄積されたプログラム資産の活用や，異なるアーキテクチャのベクトル計算機への移行が困難という点である．これに対する解決策として[4),14)]，第1にシステム側で用意した関数をユーザが呼ぶ方式，第2にプログラミング言語の仕様をベクトル演算向きに拡張する方式，第3に従来言語で記述されたプログラム中から，ベクトル処理可能な部分をコンパイラが自動的に抽出し，ベクトル処理用の目的コードを生成する方式がある．

初期システムでは，標準の FORTRAN 言語を中心にしながらも，自動ベクトル化の機能が不十分のため，たとえば IF 文をベクトル化するためにシステム関数を用いたり，部分的に言語拡張機能を用いた．しかし，現在のベクトル計算機においては，自動ベクトル化機能が大幅に強化され，第3のアプローチによる方法が主流となった[9)~27)]．

本稿は，第3のアプローチである自動ベクトル化機能をもつコンパイラの現状技術を解説する．まず，2. でベクトル計算機アーキテクチャについて述べ，コンパイラに必要な機能を概説する．3. で，自動ベクトル化コンパイラの基本技術であるデータ依存解析とベクトル化手法を説明し，4. では，ベクトルコードに対する最適化手法と，高度なプログラム変換手法について紹介する．なお，ここでは対象言語として，技術計算分野で広く使われてきた FORTRAN に限定する．

2. ベクトル計算機と自動ベクトル化コンパイラ

2.1 ベクトル計算機

ベクトル計算機の基本構造について，性能上の観点からソフトウェアにおいて留意すべきことを述べる[1)~6)]．ベクトル計算機は，ベクトル化できないスカラ部分とベクトル処理のための準備処理とを行うスカラ処理ユニットと，ベクトルデータの処理を行うベクトル処理ユニットからなる．ベクトル処理ユニットは以下の特徴をもつ．

(1)　ベクトル処理とパイプライン処理

ベクトル計算機では，ベクトルレジスタとベクトル演算器があり，**図-1**(a)の DO ループは，コンパイラにより**図-1**(b)のようなベクトル命令が生成される．これは，ベクトルデータ $\boldsymbol{B}\{B(I),\ I=1,\cdots,\ N\}$ と $\boldsymbol{C}$ が，それぞれベクトルレジスタ VR0，VR1 に一度に格納され (VL 命令)，この二つのベクトルデータが，加算器 (VADD 命令) に入力され，結果がベクトルレジスタ VR2 に求まり，これがベクトルデータ $\boldsymbol{A}$ の位置に格納 (VST 命令) されることを示している．ここで，N をベクトル長と呼び，そのベクトル命令で処理する要素数を表す．N が，ベクトルレジスタ長を超えるときは，これらのベクトル命令列を何回か繰り返して実行させる．これをストリップマイニング (strip-mining) と呼ぶ．それぞれのベクトル命令は，パイプライン方式によって高速に実行され

```
      DO 10 J=1, N          VL      VR0, B
10    A(J)=B(J)+C(J)        VL      VR1, C
                            VADD    VR2, VR0, VR1
                            VST     VR2, A
```

(a) プログラム　　(b) ベクトル命令列

図-1 FORTRAN プログラムとベクトル命令列

† Compiling Techniques for Vector Computers by Yoshikazu TANAKA and Kyouko IWASAWA (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

る．すなわち，ベクトル演算器の中はステージと呼ばれる機能単位に分けられ，データが各ステージを通過していくうちに演算が行われる．ステージ数は，計算機や演算の種類に依存する．これらのパイプラインは加算パイプライン，乗算パイプラインなど複数存在する．また，最近の日本のベクトル計算機では，一つの演算パイプラインで複数の要素を同時に処理する方式を採用している．この数をパイプラインの多重度と呼ぶ．パイプライン演算器のスタートアップ時間と，最初のデータに対する演算時間に要する時間をαとすると，それ以後はマシンサイクルτごとに多重度m個の結果が得られることから，ベクトル長nに要する時間tは，

$$t=\alpha+(n-1)\tau/m$$

で与えられる．また，ベクトル計算機の速度でよく用いられる一秒間で実行される浮動小数点数の値Mflops (Million floating operations per second) で表せば，tをμsec単位として，

$$\text{Mflops 値}=n/t$$
$$=m\tau^{-1}(1-n^{-1}+\alpha mn^{-1}\tau^{-1})^{-1}$$

となり，$n\to\infty$のとき，$m\tau^{-1}$になる．日立のS-820/80計算機では，$m=4$, $\tau=4$ nsであり，パイプライン1本あたり最大1Gflopsとなり，全体で3本の演算パイプラインが並列に動作可能なため，最大3Gflopsの演算速度が実現されている．**図-2**に，S-820におけるベクトル長をパラメータとした性能実測例を示す[7]．

（2） チェイニング

一般に各パイプラインにおいて，ベクトルデータの最初の要素の結果が出力されると，全ベクトルデータの演算の終了を待たずに，次のパイプラインを起動するチェイニングと呼ばれるデータフロー的制御が行われている．このため，図-2の例に示したプログラムの実行のタイミングチャートは，ベクトル長の大きい場合はパイプラインのスタートアップ時間が無視できるため，**図-3**のように近似的に書くことができる．ここでは，ベクトルロードとストア兼用のパイプラインが2本，ベクトル乗算及び加算のパイプラインをそれぞれ1本もつ場合の，各パイプラインの実行の様子を上から順に直線で示してある．また，チャイム(chime)[5]とは，スタートアップ時間を無視したベクトル命令の実行時間を示すもので，ベクトル長に等しいマシンサイクル数とほぼ等しい．

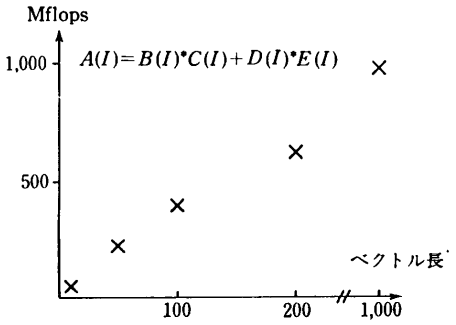


図-2 ベクトル長に対するベクトル性能の例

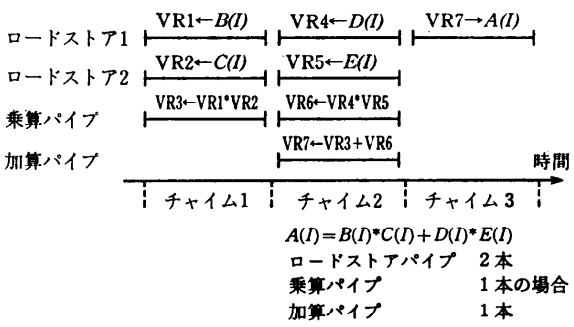


図-3 ベクトル命令の実行タイムチャート

（3） 実効性能と性能指標

ベクトル計算機の実効性能をあげる性能指標をソフトウェアからみると次の2点となる．すなわち，ベクトル化率の向上，及びベクトル加速比の向上である．ベクトル化率αは，汎用計算機で実行した場合の全実行時間のうち，ベクトル化できる部分の割合を示したもので，ベクトル加速比kとは，ベクトル化された部分がベクトル計算機で高速比される倍率である．この指標により，プログラム全体のベクトル計算機を採用したことによる性能向上率は，

$$1/\{\alpha/k+(1-\alpha)\}$$

となる．この式は，$\alpha=50\%$では，$k=\infty$であっても高々2倍にしかならず，ベクトル化率が重要であることを示し，高いベクトル化率を達成した場合にはベクトル加速比の向上が重要であることを示している．

ベクトル加速比を向上させるための基本的事項は，図-2からも分かるようにベクトル長の増大と，パイプラインの利用率の向上である．たとえば，図-3においてチャイム1は加算パイプラインが，チャイム3はロードストアパイプライン2と乗算パイプラインと加算パイプラインが使用されていない．しかし，この

例のプログラムの文と因果関係のない文がある場合は，ベクトル命令を並べ変えてこの空き時間を利用することができる．これをパイプラインの利用率の向上という．また，この図のように，一般のプログラムでは，必要なロードストアパイプラインが多いため，乗算や加算の演算パイプラインを有効に動かすことができないため，ロードストアパイプラインが性能を決めることが多い．そこで，ベクトルレジスタを有効に利用し，ロードストア回数を減少させた場合の性能をスーパベクトル性能と呼ぶことがある[6]．

2.2 自動ベクトル化コンパイラの構成

一般にスカラ計算機用のコンパイラの構造は，次のフェーズからなる．すなわち，原始言語の構文を解析し，中間コードを生成する字句解析／構文解析部，目的プログラムを効率化し，必要記憶容量を削減するための最適化部，及びデータの主記憶上の配置や，レジスタ割当てを行い目的コードの生成を行うコード生成部である[8]．ベクトル化機能を有するコンパイラでは，さらに次の機能が従来の最適化部とコード生成部に付加される[3),14),24)]．すなわち，

a．ベクトル化可否判定のためのデータ依存解析部

b．ベクトル化のため，スカラ中間コードからベクトル中間コードへ変換するベクトル化部

c．ベクトル加速比向上のためのプログラム変換部

d．ベクトルレジスタ割当て部

である．ここでは，d．に関しては，ベクトルレジスタの構成がマシン依存な部分の多いことから触れないことにする．

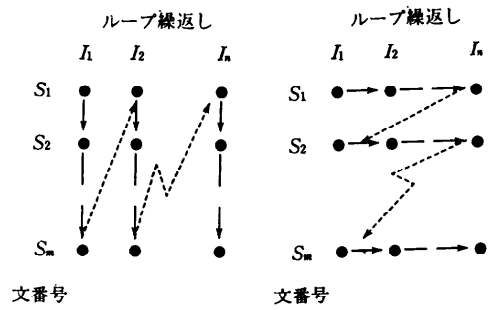
3. データ依存解析とベクトル化技術

3.1 ベクトル化の条件

DO ループ内に複数の文 $S_1, S_2, \cdots, S_m$ が順にあるとき，スカラ実行とベクトル実行では，図-4 のように文の実行順序が異なるため，同一データ要素の二つの出現に対しアクセスする相対的順序が問題となる．ループ制御変換 $I=I_k$ における文 S_i の実行を (S_i, I_k) で表現すると，同一要素に対して (S_i, I_k), (S_j, I_l) で，(使用一使用) 以外の参照 (つまり，定義一定義，定義一使用，使用一定義) がされたときに，データ参照が衝突すると言う．(i, j), (k, l) の大小関係により次の場合がある[4),9)]．

(a) $i<j$, $l<k$ または $(j<i,\ k<l)$

$i<j$, $l<k$ の場合，スカラ処理では，(S_j, I_l) の後，(S_i, I_k) を実行するのに対し，ベクトル処理では，



(a) スカラ処理 (b) ベクトル処理

図-4 ループの演算順序

```
      DO 10 J=1, N
      A(J)=B(J)+C(J)
10    C(J+1)=A(J)+D(J)
```

図-5 ベクトル化不能なプログラム例

```
      DO 10 J=1, N
      A(J+1)=B(J)+C(J)
10    D(J)=A(J)+E(J)
```

図-6 ベクトル化可能なプログラム例

```
      DO 10 J=1, N                    DO 10 J=1, N
      T=B(J)+C(J)          ⇒          $T(J)=B(J)+C(J)
10    A(J)=T+1              10        A(J)=$T(J)+1
                                      T=$(N)
```

図-7 単純変数を含むベクトル化

(S_i, I_k) の後，(S_j, I_l) が実行されるため，このままではベクトル化できない．$j<i$, $k<l$ の場合も同様にベクトル化できない．図-5 の配列 C がこの関係となる．

(b) その他の場合

図-6 の配列 A のように，参照順序関係は同一なので，ベクトル化可能である．

単純変数やループ内で添字が不変な配列における (使用一使用) 以外の2出現は，同一のアドレスを参照するので衝突が発生し，かつデータ参照関係がベクトル化に適さない．しかし，これらの変数に対して図-7 のようにコンパイラが新1次元配列 (実際にはベクトルレジスタ上に置かれる) に置き換えること，及びループ終了後の終値保証処理によりベクトル化できることが多い[11),17)]．

配列のデータ参照の衝突の有無は次のように調べる．すなわち，(S_i, I_j) に出現する k 次元配列の添字の組を $(f_1, f_2, \cdots, f_k)$, $(S_{i'}, I_{j'})$ に出現する添字の組を $(f_1', \cdots, f_k')$ で与える．ただし，f_p は，I_j の関数であり，f_p' は，$I_{j'}$ の関数である[4),9),14)]．また，多重ルー

プのベクトル化のための場合の解析においては，それぞれのループの制御変数の関数である[3),10),20),27)]．参照の衝突が存在する可能性があるのは，次の

$f_1=f_1', \cdots, f_k=f_k'$

Diophantine 方程式が，制御変数として有効な整数解をもつ場合である．図-6 の配列 A の場合，次の方程式を解けばよい．

$j+1=j'$, $1 \leqq j \leqq N$, $1 \leqq j' \leqq N$

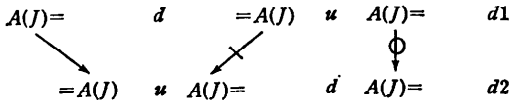
この整数不定方程式は，簡単に解が得られる．しかし，一般に多重ループ内の場合は，添字に定数以外にはループ制御変数のみが使われる場合でも，解析能力が十分でない場合がある．図-8 にその例を示す．これは，整数不等式を不等式の変形で進めるため，必要条件でしか検査できないためである．これに対して，津田ら[3),19)]は，コンパイル時に多重ループ全体のループ回数が決まる場合（図-8 の例では，ループ回数 50），添字値の数例をコンパイル時に生成しソートしてマージする方法を併用して解決を行った．

3.2 データ依存グラフとベクトル化

Kuck[11)]らは，ベクトル化のためのプログラム変換が可能かどうかを決定する際に，データ依存グラフを用いる方法を示した．データ依存グラフとは，各変数参照の頂点を次の 3 種のデータ依存関係をあらわす辺で結んだ有向グラフであり，実行順序関係の制約を表現する．データ依存関係には，図-9 のように，変数定義 d における値を変数使用 u において使用する可能性のあるフロー依存，変数使用 u における値が変数定義 d で書き換えられる可能性がある逆依存，変数定義 $d1$ における値が変数定義 $d2$ で書き換えられる可能性のある出力依存がある．ここで，逆依存と出力依存は，単一代入性をもつ言語では表現できない人工的なものである．これらの依存は，図-10 のようにリネームや配列化によるプログラム変換で除去[11),12)]可能な

```
      DO 10 I=1, 2
      DO 10 J=1, I+1
      DO 10 K=1, 2*I+J+5
      A(2*I-J+5)=1
10    A(2*K+I+2)=2
```

図-8 多重ループの例



(a) フロー依存　(b) 逆依存　(c) 出力依存

図-9 データ依存関係の種別

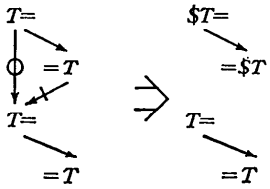
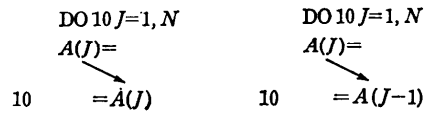
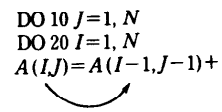


図-10 逆依存，出力依存のリネームによる除去



(a) ループ独立依存　(b) ループ運搬依存

図-11 ループ依存性による依存種別



(a) ループ依存性による表現
DO 10 に関するループ運搬フロー依存
(DO 20 に関しては依存なし)

(b) 方向ベクトルによる表現
方向ベクトル (<, <) をもつフロー依存

図-12 ループ付加情報のもち方

場合がある．

以上で述べたデータ依存は，DO ループの繰り返しに関する依存に，さらに細かく分類できる．これには，次の二つの方法がある．第 1 は，図-11 に示すようにループの 1 回の実行の間で生じるループ独立依存と，ループを 2 回以上実行することにより生じるループ運搬依存にわける方法である[18),21),25)]．第 2 は，図-12(b)に示すように，各ループインデクスごとのループ繰り返しに関する依存相対関係（<，>，=の記号で示す）を独立に求め方向ベクトルとして付加する方法である[12)]．両者の表現法の比較を図-12 のプログラムを用いて行う．プログラム上の配列 A は，ループインデクス ($I=i$, $J=j$) で定義した値をループインデクス ($i+1$, $j+1$) で使用するフロー依存であるが，第 1 の方法では，DO 20 のループの繰り返しでは依存が発生せず，DO 10 のループの繰り返しで初めて発生する依存であるため，DO 10 に関するループ運搬依存と定義される．それに対し，第 2 の方法では，依存のあるループインデクス (i, j) $(i+1, j+1)$ 間の相対的関係から，方向ベクトル (<, <) で表現される．

これらの依存グラフは，配列参照の添字の比較を行

い直接求めることができる[10),12)]．しかし，すべての2出現参照の比較による方法だけでは，たとえば，**図-13** で示したにせの依存が付加されるため，最適化において障害になることがある．そこで，金田ら[21),22)]は，変数の到達定義や露出使用を求める大域的データフロー解析[8)]と組み合わせることにより解決を行った．

3.3 ベクトル化の方法

前節で説明した依存グラフをもとにベクトル化のための変換方法を述べる[4),11)]．

a．依存グラフに強連結成分があるかどうかを調べる．

b．強連結成分がない場合，グラフの有向辺に従ったトポロジカルソートにより，対応文を並べかえればベクトル化できる．

c．強連結成分がある場合で，強連結を構成する有向辺の少なくとも一つの辺が逆依存または出力依存の場合，始点オペランドを別の文として切り出すと強連結が解消されるため，bの処理を行えば良い．これをノード分割と呼ぶ．

d．強連結成分がある場合で，強連結を構成する有向辺がすべてフロー依存の場合は再帰的参照であるため，この部分は本質的にベクトル化できない．

図-14 にベクトル化のためのグラフ変換の例を示してある．ここでは，文 S_3 のオペランド $C(J+2)$ がノード分割され，再帰的関係である S_1, S_2 以外はベクトル化できる関係であることが分かる．**図-15**(a)にベクトル化された結果を示す．ここで示したように，ループを複数のループに分けることは，ベクトル

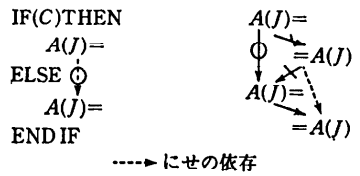


図-13 にせの依存の検出

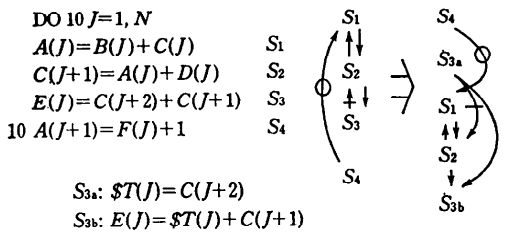


図-14 ベクトル化のためのグラフ変換

```
A(2: N+1)=F(1: N)+1          S4
$T(1: N)=C(3: N+2)           S3a
  DO 11 J=1, N
  A(J)=B(J)+C(J)             S1
11 C(J+1)=A(J)+D(J)          S2
E(1: N)=$T(1: N)+C(2: N+1)   S3b
```

(a) 通常ベクトル命令による自動ベクトル化変換

```
A(2: N+1)=F(1: N)+1          S4
$T(1: N)=C(3: N+2)           S3a
$T2(1: N)=B(1: N)+D(1: N)    S1a
C(2: N+1)=C(1: N)+$T2(1: N)  S2b
A(1: N)=B(1: N)+C(1: N)      S1
E(1: N)=$T(1: N)+C(2: N+1)   S3b
```

(b) マクロベクトル命令を使用した自動ベクトル化変換

図-15 自動ベクトル化変換（図-14 のプログラム例）

ベクトル和	$S=S+A(J)$
内積	$S=S+A(J)*B(J)$
最大	IF($S.LT.A(J)$)$S=A(J)$
最小	IF($S.GT.A(J)$)$S=A(J)$
一次再帰演算	$A(J+1)=A(J)*B(J)+C(J)$

図-16 マクロベクトル命令の例

化において最も基本的な変換[13)]と言える．

ところで，再帰的な関係でベクトル化できない場合でも，特別のハードウェアサポートがあればできる場合がある．たとえば，現在のベクトル計算機では，**図-16** のようなマクロ機能のベクトル命令をサポートしていることが多い．これらの演算では，変数 S，一次再帰演算の例では配列 A が強連結を構成しているが，たとえば，ベクトル和に関しては一連の要素を加算する演算であるので，特別のアーキテクチャ上の工夫によりベクトル命令が実現できる．従来，コンパイラはこれらの演算をパターンマッチング[17)]により検出していたため，プログラムの書き方によっては検出できない場合があった．これに対し，筆者ら[25)]は再帰計算部を標準形に変換することにより検出率を高めた．**図-15**(b)に，図-14 の再帰部に一次再帰演算ベクトル命令（S_{2b} の部分）が適用され，全体がベクトル化される例を示す．

4. ベクトルコード最適化手法

ベクトルコードの最適化は，共通式の削除といった汎用的な最適化と，特にベクトル性能向上を意識したプログラム構造の再構成による二つに分けられる．

4.1 汎用的最適化

汎用的最適化とはスカラコード／ベクトルコードに共通的な最適化である．この中でベクトルコードの場合，特に重要であるのは，配列オペランドの共通化を行いレジスタ上に値を保持する最適化である．なぜな

ら，2.で述べたように，ベクトル計算機では主記憶への負荷がプログラム実行性能のボトルネックとなりやすいためである．図-17 にいくつかの例を示す．ここで，変数Tは通常ベクトルレジスタに保持されることを意味する．(a)の例は配列の定義一使用，(b)の例は使用一使用の例である．(a)の場合，配列Aに関して共通化できなければ，ベクトルロード命令が余分に必要となるばかりでなく，複数ロードストアパイプラインを有する計算機では，主記憶参照順序関係を保つため，ストアとロード命令間に主記憶シリアライズ命令を入れるか，またはストアとロードを同一パイプラインに指定する必要があり，複数パイプラインの並列実行ができない．(c)の最適化はコード引上げ (code hoisting)[8] と呼ばれる技法である．この最適化は，スカラではコードの大きさを削減するにすぎない．しかし，ベクトル計算機では条件節の真偽両方に対してマスク付きで主記憶参照を行うため，この最適化は主記憶へのアクセスを減らすことになり，実行時間も削減できる．

4.2 ベクトル計算機向けプログラム変換

ベクトル計算機の性能を十分に引き出すためのプログラム変換としてコンパイラが自動的に行うものとして，多重ループのループ交換，多重ループの一重化，多重ループのループアンローリングがある．

(1) 多重ループのループ交換

これは，図-18 のように，最内ループでは再帰的関係でベクトル化できないが，外側ループに関してベクトル化できる場合や，どちらのループでもベクトル化可能なときに，たとえば外側ベクトル長が長く効率的と判断した場合に，ループを交換してベクトル化する手法である．ループ交換できるための十分条件の一つは，図-18 の依存グラフによる変換過程を見れば分かるように，以下のように述べることができる．

「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とき，最内ループと外側ループは交換可能」

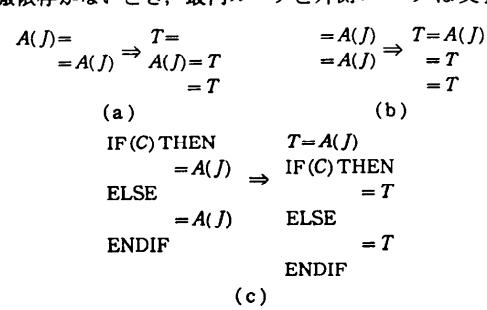


図-17 ベクトルコードの最適化

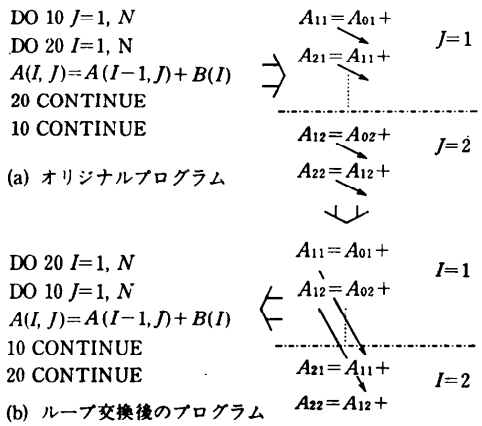


図-18 多重ループのループ交換

しかし，この十分条件では実用上は弱いため，図-12のように外側ループにループ運搬性依存が存在するときは，ループに関する情報として，方向ベクトルを用いた判定法もある[12]．

(2) 多重ループの一重化

これは，多重ループで，各ループの制御変数による配列の添字アクセスが連続となる場合，図-19(a)のようにループを一重化し，ベクトル長を増大させることにより，ベクトル実行効率をあげる方法である．この条件による変換は，配列の初期化などに狭い範囲しか適用できない．これに対し，間接アドレス方式を用いて多重ループインデクスを一重化するプログラム書法が知られていたが，津田ら[19]は，多重ループ全体の処理数が，図-19(b)のようにコンパイル時に確定する場合には，コンパイル時に間接アドレスをデータ文のように生成する自動化方式を実現し，適用の範囲を

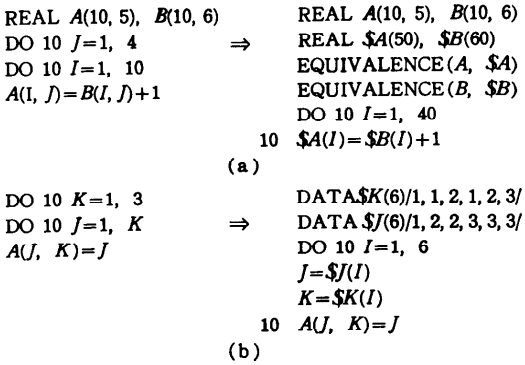


図-19 多重ループの一重化

広げた.

(3) 多重ループに関するループアンローリング

ループアンローリングとは，複数並列パイプラインの並列動作が十分に行われるように，代入文の右辺の演算量を増し，同時に実行されるベクトル演算を増加させる方法である．たとえば，$N\times N$ の正方行列の乗算プログラムは図-20(a)のように書くことができるが，これを図-20(b)のように，ベクトル化ループ長を変えずに外側ループに関して展開する方法である．例では展開数を2としたが，最適な数はパイプラインの数や演算のパターンに依存する．

また，外側ループアンローリングが適用できるためには，データ依存関係を満足する必要がある．なぜなら，ベクトル化された部分に対しては，ベクトルレジスタ長ごとの繰り返し処理（ストリップマイニング）ループが生成されるため，元々のベクトル長がベクトルレジスタ長以下であることが分からないかぎり，ループ交換と同様な適用条件が必要である．

(4) プログラム変換による性能実例

行列乗算を例に，日立のベクトルコンパイラによるS-820の性能を示す．表-1が，行列乗算の種々のループインデクスの順番指定によるコーディング書法

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

(a)

⇒

```
      DO 20 J=1, N
      DO 11 K=1, N-1, 2
      DO 11 I=1, N
        A(I, J)=A(I, J)+
          B(I, K)*C(K, J)+
11        B(I, K+1)*C(K+1, J)
      IF(MOD(N, 2). NE. 0) THEN
        DO 12 I=1, N
        A(I, J)=A(I, J)+
12            B(I, N)*C(N, J)
      ENDIF
20    CONTINUE
```

(b)

図-20 行列乗算プログラムのアンローリング

表-1 最適化手法の違いによる性能差
使用計算機 日立 S-820/80
コンパイラ FORT 77/HAP V23-0 G

プログラム書法	最適化レベル OPT 3 (Mflops)	最適化レベル SOPT (Mflops)
内積 JIK タイプ	688	802
中間積 JKI タイプ	802	1661
外積 KJI タイプ	802	1540

行列のサイズ 512×512

```
      DO 20 J=1, N
      DO 50 II=1, (N-1)/VLG+1
      IS=(II-1)*VLG+1
      IE=MIN(N, II*VLG)
      DO 51 I=IS, IE
51    VR=A(I, J)
      DO 10 K=1, N
      DO 10 I=IS, IE
10    VR=VR+B(I, K)*C(K, J)
      DO 52 I=IS, IE
52    A(I, J)=VR
50    CONTINUE
20    CONTINUE
```

図-21 多重ループにわたるベクトルレジスタ保持のための変換

とコンパイルのオプション（最適化レベルの指定）による性能差である．中間積とは，図-20(a)のようにループ制御変数が外から J, K, I の場合で，内積とは，J, I, K の場合，外積とは K, J, I の場合である[3),4)]．また．配列 A に対する初期設定文は，適切な場所に入ったもので測定した．OPT 3 の場合は，多重ループに対する変換は行わないので，プログラムに書いたとおりのベクトル化を行い内積形は内積タイプのベクトル命令で，その他は積和タイプのベクトル命令が使用される．これに対し SOPT では，多重ループに対する最適化を行う．すなわち，内積タイプのコーディングに対しては，内積より積和ベクトル命令が高速と判断し，Kループと Iループを交換して中間積タイプと同一の形になる．ただし，アンローリングは適用されていない．中間積と外積タイプに対しては，8倍のアンローリングが適用されている．さらに，中間積タイプでは，図-21で示すように多重ループに対してベクトルレジスタ上に $A(I, J)$ を保持するためのプログラム変換が行われている．この結果，最内ループ内のオペレーションの比は，ロードパイプ：乗算パイプ：加算パイプ＝1：1：1 で主記憶への負荷が減っているため，いわゆるスーパベクトル性能が実現されている．この図で，VLG はベクトルレジスタ長，VR はベクトルレジスタを意味している．

5. おわりに

以上，ベクトル計算機に対する自動ベクトル化について，その方式を紹介した．

今後の課題としては，FORTRAN 以外の他言語への拡張があげられる．他言語，たとえば Pascal, C などへの拡張を考えるためには，FORTRAN の DO ループに対応する for 文など以外のループ構造に対するベクトル化を考える必要がある．たとえば，while

ループが重要である．このループでは，一般にループ脱出条件がループ実行中に決まるため，これに対する解決が必要である．また，構造体に対するベクトル化も，他分野に対する応用を広げることになるため重要な課題と考えられる[3]．

参考文献

1) Fernbach, S. ed. (長島重夫訳)：スーパーコンピュータ，p. 343，パーソナルメディア (1988).
2) 村田健郎，小国 力，唐木幸比古：スーパーコンピュータ，p. 304，丸善 (1985).
3) 津田孝夫：数値処理プログラミング，p. 366，岩波書店 (1988).
4) 島崎眞昭：スーパーコンピュータとプログラミング，p. 238，共立出版 (1989).
5) Levesque, J. M. and Williamson, J. W.: A Guidebook to Fortran on Supercomputers, p. 218, Academic Press (1989).
6) Dongarra, J. J. and Eisenstat, S. C.: Squeezing the Most out an Algorithm in CRAY FORTRAN, ACM Trans. Math. Softw., Vol. 10, pp. 219-230 (1984).
7) Eoyang, C., Mendez, R. H. and Lubeck, O. M.: The Birth of the Second Generation: The Hitachi S-820/80, Proc. Supercomputing '88, IEEE, pp. 296-303 (1988).
8) Aho, A. V. and Ullman, J. D.: Principles of Compiler Design, p. 604, Addison-Wesley (1979).
9) Takanuki, R., Nakata, I. and Umetani, Y.: Some Compiling Algorithms for an Array Processor, Proc. 3rd USA-Japan Computer Conference, pp. 273-279 (1978).
10) Banerjee, U.: Speedup of Ordinary Programs, Ph. D. Thesis, Dept. of Comp. Sc., Univ. of Ill. at Urb.-Champ. (1979).
11) Kuck, D. J., Kuhn, R. H., Padua, D. H., Leasure, B. and Wolfe, M.: Dependence Graphs and Compiler Optimizations, Proc. 8th Annual ACM Symposium on Principles of Programming Languages, pp. 177-189 (1981).
12) Wolfe, M. J.: Optimizing Supercompiler for Supercomputers, Ph. D. Thesis, Dept. of Comp. Sc., Univ. of Ill. at Urb. -Champ. (1982).
13) 安村通晃，梅谷征雄，堀越 彌：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，情報処理学会論文誌，Vol. 24, No. 1, pp. 15-21 (1983).
14) 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).
15) Kamiya, S., Isobe, F., Takashima, H. and Takiuchi, M.: Practical Vectorization Techniques for the FACOM VP, Information Processing 83, North Holland, pp. 389-394 (1983).
16) Umetani, Y. and Yasumura, M.: A Vectorization Algorithms for Control Statements, J. Inf. Process., Vol. 7, No. 3, pp. 170-174 (1984).
17) Yasumura, M., Tanaka, Y., Kanada, Y. and Aoyama, A.: Compiling Algorithms and Techniques for the S-810 Vector Processor, Proc. '84 International Conference on Parallel Processing, pp. 285-290 (1984).
18) Allen, J. R. and Kennedy, K.: Automatic Loop Interchange, Proc. of the ACM SIGPLAN '84 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 19, No, 6, pp. 233-246 (1984).
19) Tsuda, T. and Kunieda, Y.: Mechanical Vectorization of Multiply Nested DO Loops By Vector Indirect Addressing, Information Processing 86, North Holland, pp. 785-790 (1986).
20) 石田和久，金田 泰：多重ループの配列多重添字解析方式，情報処理学会プログラミング言語研究会，87-PL-10 (1987).
21) 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).
22) Gotou, S., Tanaka, Y., Iwasawa, K., Kanada, Y. and Aoyama, A.: Advanced Vectorization Techniques for Supercomputers, J. Inf. Process., Vol. 11, No. 1, pp. 23-31 (1987).
23) Shimasaki, M.: On Techniques in Vectorizing Compilers and Optimizing Program Transformations for Supercomputers, J. Inf. Process., Vol. 11, No. 1, pp. 2-14 (1987).
24) Allen, R. and Kennedy, K.: Automatic Translation of FORTRAN Programs to Vector Form ACM Trans. Prog. Lang. Syst., Vol. 9, No. 4, pp. 491-542 (1987).
25) Tanaka, Y., Iwasawa, K., Gotou, S. and Umetani, Y.: Compiling Techniques for First-Order Linear Recurrences on a Vector Computer, Proc. Supercomputing '88, IEEE, pp. 174-181 (1988).
26) Tsuda, T. and Kunieda, Y.: V-Pascal: an Automatic Vectorizing Compiler for Pascal with No Language Extensions, Proc. Supercomputing '88, IEEE, pp. 182-189 (1988).
27) Girkar, M. and Polychronopoulos, C.: Compiling Issues for Supercomputers, Proc. Supercomputing '88, IEEE, pp. 164-173 (1988).

(平成2年1月26日受付)