プログラミング言語　25－2
（１９９０．６．８）

関数型言語ＦＰの
ベクトルプロセッサ向き自動変換法

武宮博
（日立東北ソフトウェア）
布川博士　白鳥則郎　野口正一
（東北大学電気通信研究所）

本稿では，スーパコンピュータのベクトル処理を表現するのに適した言語として関数型言語FPをとりあげ，FPで書かれたプログラムをFORTRANへ自動変換する手法を述べる．FPの関数の中でベクトル処理に適したものを選び，それらの組み合わせによって生成される関数は全てベクトル処理されるように変換することを目的としている．その結果，プログラマ及びシステムはデータ依存解析を行なうことなくプログラム中のベクトル処理可能部分を認識できる．また，それらの関数を多用することにより，自然にベクトル処理に適したプログラムを構成できる．この変換法の効果を調べるため，スーパコンピュータSX-2N上で行列積を計算するプログラムを実行したところ，実行速度の向上は３倍程度，ベクトル化率は93%であった．

The Compiling Method of The Functional Language FP for Vector Processing

Hiroshi Takemiya
Hitachi Tohoku Software Co.,Ltd
Hiroshi Nunokawa,Norio Shiratori,Syoichi Noguchi
Reserch Institute of Electrical Communication,Tohoku University

Abstract

The functional language FP is powerful for parallel processing.To carry out vector processing of FP program,we propose to translate the FP program into FORTRAN .This method can generate highly vectorized FORTRAN programs for all combinations of vector processable FP functions.As a result,both programmer and computer can recognize vector processable parts in the program without analyzing data dependence relationship.We executed matrix multiplication program in FP on the supercomputer SX-2N.It excuted nearly 3 times faster than in the sequential mode and its vectorization ratio is 93 percent.

1. はじめに

現在のスーパコンピュータはベクトル演算機構を装備しており，同一操作を多数のデータ（ベクトルデータ）に作用させる場合，極めて高い処理能力を持つ．しかし，一般ユーザ向けに開放されている言語がFORTRANしかないため，その利用は数値処理に限られている．そこで，近年スーパコンピュータの持つ高い処理能力を記号処理など数値処理以外の分野に利用しようという研究が行なわれている．([1],[2],[3],[4])

本稿では，ベクトル演算に適した言語として関数型言語FPを取り上げる．FPのコンストラクタのうち何がベクトル処理の対象となるかを考え，それらを全てベクトル化できるようにFORTRANへ自動変換する手法を論じる．このように，明確にベクトル化の効果のあるコンストラクタを示すことによって，ユーザ及びシステムはプログラムレベルでベクトル化可能部分を認識できるようになる．

この変換手法を用いて行列の積を求めるFPのプログラムを実際にスーパコンピュータ上で実行させ，ベクトル化の効果を測定した．

2. FPプログラムのベクトル化

2.1 FORTRANプログラミングの問題点

現在スーパコンピュータ上ではFORTRANプログラムがベクトル実行されており，ベクトル実行の対象となるのは，DOループである．

しかし，FORTRANのDOループは逐次処理と並列処理の両方を表わしており，並列処理可能であるかどうかを知るには，ループ内のデータ依存関係を調べなければならない．これは，DOループがもともと逐次的繰り返し処理を表記するためのコンストラクタであったためであり，並列処理という従来の計算機にはなかった新しい概念をFORTRANで表記しようとする際の障害となっている．

この点を改善するために，コンパイラによってベクトル実行可能な部分を自動認識するためのデータ依存関係解析手法が提唱されているが([5])，リストベクトルを使用するようなプログラムではプログラマがコンパイラディレクティブによってベクトル化を指示しなければならない．

一方，関数型言語FPでは，並列処理を自然に表現できる関数が存在する．従って，プログラマが意図した並列処理を処理系がデータ依存解析を行なうことなく認識できる．

現在開発中のFP自動変換システムでは，FPの関数の中で並列処理(ベクトル処理)を表現するものは何かを考え，それらを全てを自動的にベクトル実行できるようにFORTRANへ変換することを目的としている．これによって，処理系及びプログラマの両方がデータ依存解析を行なう事なくプログラム上で並列実行可能な部分を認識できる．

2.1 変換の留意点

FPの関数には，

①原始関数

②プログラム構成子（・，α，[]，/，等）によって関数を組み合わせた関数形式

③関数を定義し名前を付けた関数定義

がある．([6]) （現段階では，定義された関数（再帰関数を含む）の処理は考慮していない）

これらの関数の内，

①原始関数でベクトル化の効果があるもの:pri_v

②プログラム構成子でベクトル化の効果があるもの:ff_v

と表記すると，上記の関数の組み合わせである

pri_v

$ff_v(pri_s)$

$ff_v(pri_v)$

$ff_v(ff_s(pri_s))$

$ff_v(ff_v(pri_s))$

を，全て効率良くベクトル処理できるように変換することが必要となる．

[ここで，pri_s,ff_sは，スカラ処理を行なう原始関数及び　関数形式を表わす]

上記の組み合わせを，ベクトル処理できるようにする際には，以下の点を考慮しなければならない．

①記憶域への連続アクセス

スーパコンピュータでは，記憶域へのアクセス方式としてバンクインタリーブ方式を採用しており，一度に多数のバンクを通して，データを（ロード／ストア）できる．しかし，不連続アクセスの場合はバンクコンフリクトが生じ，性能が極端に落ちる場合がある．したがって，なるべく連続アクセスのできるようなデータ構造を採用する必要がある．

②ベクトル化可能部分の増大

DOループで記述できるにもかかわらず，ベクトル化できない要因としてdata dependence loopの存在があげられる．([7]) データ依存関係には，flow,anti,outputの3種類の依存性があるが，この内flow dependence loopは除去できない．しかし，外側にベクトル化可能なDOループがかかったときにはloop interchangeによってベクトルが可能となる．

従って，なるべくdata dependence loopを生じさせないように関数の作用を記述し，また，本質的にdata dependence loopが生じてしまうような逐次実行型の関数に対しては，その関数を複数のオブジェクトに作用させる際，loop interchangeが行なわれるように関数の作用を記述する．

③ループの連続化

ベクトル演算は，演算開始前に前処理が必要であり，一定の立ち上がり時間を必要とする．従って，ループ長の等しいループはなるべくひとまとまりにしたほうが立ち上がりにかかる時間が少なくて済む．また，命令が並列実行される機会も増える([8])．よって，ループ長の等しいループはなるべくひとまとまりになるように関数を記述する必要がある．

以上の点を考慮して，効率良くFPプログラムをFORTRANへ自動変換する手法を以下で述べる．

2.2 FPプログラムのベクトル化手法

前述の関数の組み合わせを全てうまくベクトル化するには，データ構造，原始関数及びプログラム構成子の動作の規定が重要である．これらを順に述べていく．

2.2.1 データ構造の規定

今回のFP自動変換システムでは，列，整数，真偽値の3種類のオブジェクトを取り扱う．

オブジェクトは図1のような形で内部に保持される．オブジェクトはリストで表現され，リスト構造の深さに応じてセルを割り当てる．リストの構成要素が列になっている場合はその列へのポインタを代入する．整数あるいは真偽値になっている場合はその値を代入する．

HP arrayには，関数の作用対象のアドレスを示すポインタが格納される．オブジェクトの下部構造に作用する関数形式の適用時には，プログラム構成子の動作によって，HP arrayのIあるいはJ方向に下部構造のアドレスを示すポインタが格納される．また，関数の作用対象が整数/真偽値の場合は，それらの値そのものがHP arrayに登録されることになる．HPLENにはHP arrayに登録されたポインタの個数が，BACKADにはHP array内のポインタの登録元のアドレスが格納される．

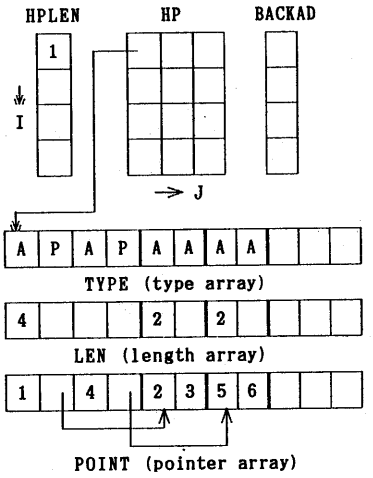

図1 データ構造

このようなデータ構造にすることによって，ポインタや整数/真偽値のロード,セーブが連続的に行なえるため，高速なベクトル処理が期待できる．

2.2.2 原始関数の規定

原始関数は，HP array に登録された関数適用対象に対して定義に従って作用する．ただし，並列処理を意識して，作用対象が１つであるか複数個あるかによって，生成されるコードは異なる．

基本的には原始関数の適用は元のセルに結果を格納するよう記述する．これは次のような理由による．

原始関数の適用結果を新しいセルに格納すると，未使用領域を示すポインタNEWPを関数の適用ごとに更新しなければならない．この原始関数を複数の対象に作用させる場合には，NEWPの更新部分のためにデータ依存性が生まれ，ベクトル実行ができなくなってしまうためである．

原始関数の作用自体にベクトル実行可能性があるのは，以下の関数である．

TRANS,APND,DIST,ROT,REVERSE,IOTA,CONCAT:PRI_V

これらの関数は，作用をFORTRANで記述するだけでベクトル実行される．

FP自動変換システムでは，上述のデータ構造を生かしたPRI_Vとして，VADD，VSUB，VMULT，VDIVを組み込んである．それぞれの定義は

・VADD

$VADD:X \equiv X=<<X_1,\cdots,X_N>,<Y_1,\cdots,Y_N>>$

$\rightarrow<x_1+y_1,\cdots,x_n+y_n>;\perp$

ただし，X_i,Y_iは数

・VSUB

$VSUB:X \equiv X=<<X_1,\cdots,X_N>,<Y_1,\cdots,Y_N>>$

$\rightarrow<x_1-y_1,\cdots,x_n-y_n>;\perp$

ただし，X_i,Y_iは数

・VMULT

$VMULT:X \equiv X=<<X1,\cdots,XN>,<Y1,\cdots,YN>>$

$\rightarrow<x1\times y1,\cdots,xn\times yn>;\perp$

ただし，Xi,Yiは数

・VDIV

$VDIV:X \equiv X=<<X1,\cdots,XN>,<Y1,\cdots,YN>>$

$\rightarrow<x1\div y1,\cdots,xn\div yn>;\perp$

ただし，Xi,Yiは数(Yi÷0=⊥)

定義から分かるように，これらの原始関数は記憶域から連続的にデータをベクトル演算器へ投入し，演算結果を連続的に記憶域に格納するというベクトル演算に適した関数となっている．

2.2.3 プログラム構成子の規定

プログラム構成子の中で，並列性を持つものは，

applytoall(α) :ffv

である．

プログラム構成子の動作は，作用する関数の動作を挟んで前処理と後処理に分割される．例として，α(+)の動作を図２に示す．

```
    α の前処理
DO J=1,LEN(HP(1,1))
   HP(2,J)=INTER(HP(1,1)+J-1)
   HPTYPE(2,J)=TYPE(HP(1,1)+J-1)
   BACKAD(2,J)=HP(1,1)+J-1
END DO
   HPLEN(2)=LEN(HP(1,1))
   I=2
DO J=1,HPLEN(I)
```

```
    ＋の処理
HP(I,J)=POINT(HP(I,J))+POINT(HP(I,J)+1)
INTER(BACKAD(I,J))=HP(I,J)
```

```
    α の後処理
END DO
I=1
```

図２　α(+)の動作

①αの前処理部では，+が作用すべき作用対象をHP arrayに登録する．

②+の処理部では，αの登録した作用対象に対して，+の演算をベクトル実行する．

③αの後処理部では次の関数のために，元のHPに作用対象を戻す．

このように，αが登録した作用対象に対してprisの作用をベクトル実行することによって，ffv(pris)はベクトル化されることになる．

次に，ffv(priv)のベクトル実行に関して述べる．ffv(priv)のベクトル実行に関しては，二つの困難がある．それを，privがapndlの場合について述べる．

APNDLの場合，通常生成される生成されるコードは図3のようになる．

```
*VDIR NODEP
 DO N=1,LEN(INTER(HP(I,J)+1))
  POINT(NEWP+N)=POINT(POINT(HP(I,J)+1)+N-1)   ①
  TYPE(NEWP+N)=TYPE(POINT(HP(I,J)+1)+N-1)
 END DO
-------------------------------------------------
POINT(NEWP)=POINT(HP(I,J))
TYPE(NEWP)=TYPE(HP(I,J))
LEN(NEWP)=LEN(INTER(HP(I,J)+1))+1              ②
HP(I,J)=NEWP
POINT(BACKAD(I,J))=HP(I,J)
-------------------------------------------------
NEWP=NEWP+LEN(NEWP)                            ③
```

図3　APNDLの動作

このコードでは，①の部分のDOループがベクトル実行される．（これがprivとしてのベクトル化の効果となる）

しかし，APNDLにαがかかった場合，APNDLにαのベクトル化の効果を出すことは困難である．

まず，APNDLでは，作用後のデータ格納セルの長さが作用前のセルの長さより長くなるので未使用セル領域にデータを格納しなければならない．しかし，一般にAPNDLの作用対象の長さは決まってないので③の演算が終了するまで次のHP arrayに登録された作用対象に対して②の部分の演算が開始できない．従って，②の部分をベクトル処理することはできない．また，①の部分も二重のDOループとなってαの効果はでない．

よって，APNDLにαがかかったとき，αの効果はでないことになる．

一般に，ベクトル化の効果がある原始関数(構造変換関数）にαを適用しても，APNDLと同様な理由でαの効果は出ない．

しかし，データが特殊な構造をしているときは，αの効果を出すことができる．

例えば，

α(APNDL):<<1,<2,3>>,<4,<5,6,>>,<7,<8,9>>>

の場合を考えてみる．

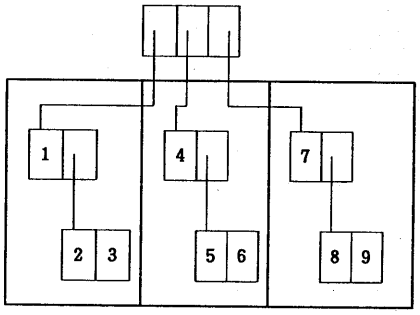

図4　APNDLの作用対象

APNDLの適用対象となるのは，<1,<2,3>>,<4,<5,6>>,<7,<8,9>の3つのオブジェクトである．各オブジェクトは，各々アトムの値が異なるだけでリスト構造は等しい．この場合，データの格納先が先読みできるので，上記プログラムの①の部分がベクトル化できる．①の部分をベクトル化したAPNDLのプログラムは図5のようになる．

図5において，①の部分は二重ループとなるためαの効果は出ないが，J,Kのどちらのループに関してもベクトル化できるので，ループ長の長い方をベクトル化対象とするようにコード生成することもできる．

このように"等構造性"という概念を導入し，プログラマがそれを指示することによって，$ff_v(pri_v)$の場合もベクトル化の効果を出すことができる．

```
 LENTEMP=LEN(INTER(HP(I,1)+1)+1
*VDIR NODEP
  DO J=1,HPLEN(K1)
*VDIR NODEP
   DO K=1,LENTEMP-1
    INTER(NEWP+(J-1)*LENTEMP+K)=                    ①
                INTER(INTER(HP(I,J)+1)+K-1)
    TYPE(NEWP+(J-1)*LENTEMP+K)=
                TYPE(INTER(HP(I,J)+1)+K-1)
   END DO
-------------------------------------------------------
   INTER(NEWP+(J-1)*LENTEMP)=INTER(HP(I,J))
   TYPE(NEWP+(J-1)*LENTEMP)=TYPE(HP(I,J))
   LEN(NEWP+(J-1)*LENTEMP)=LENTEMP
   HP(I,J)=NEWP+(J-1)*LENTEMP                       ②
   POINT(BACKAD(J))=HP(I,J)
  END DO
-------------------------------------------------------
  NEWP=NEWP+LENTEMP*HPLEN(I)                        ③
```

図 5　α (APNDL) の動作

次に，$ff_v(ff_s(pri))$ の場合のベクトル化の方式を，α (/(+)) を例にして述べる．

/の定義は

$$/f:x \equiv x=<x1,\cdots,xn> \ \&\ n \geq 2 \rightarrow f:<x1,/f:<x2,\cdots,xn>>;\perp$$

である．この作用をFORTRANで記述すると各2要素間の演算に対してデータ依存性が存在する．すなわち，最初の2要素の結果がでないと以下の演算が行なえないので，ベクトル演算は実行できない．

従って，プログラム構成子の動作の規定どおりにα(/(+)))を実行すると，/の逐次実行ループが内側ループになるため，αのベクトル実行の効果がでない．そこで，αのベクトル処理ループの効果を+の処理に反映させるために，FP自動変換システムでは，次のような処理を行なう．

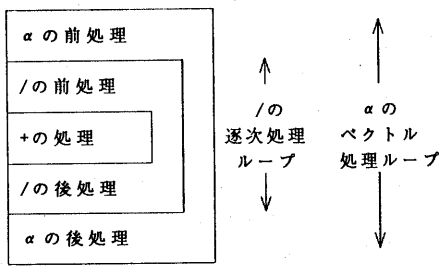

図 6　α (/(+)) の動作

①まず，αの前処理として，/(+)の適用対象をHP arrayに登録する．

②次に，/の前処理として，リストの構造を変換し，逐次処理の適用対象をHP arrayに登録する．
その際，HP arrayへの登録を縦方向に行なう．

③+の処理をHP arrayのI方向に登録されたポインタ（またはアトム）に対して行なう．I方向に登録されたポインタ（アトム）に関しては，データ依存性がないので，+の処理はベクトル実行可能である．

④/の後処理として，+の適用対象を，HP arrayの一段上に格納されたポインタに移す．

⑤/の処理対象がなくなるまで，逐次的に③，④の処理を繰り返す．

⑥最後に，αの後処理としてα起動前のポインタに適用対象を移す．

以上の操作によって，αの効果を/の作用する関数にも及ぼすことができる．

同様な操作によって，consにαがかかった場合もαの効果をconsの作用する関数に及ぼすことができる．

また，conditionにαがかかったときは，マスクベクトルを利用して，条件に適合するデータ対象と適合しないデータ対象を収拾し，それぞれHP arrayのJ方向にポインタを登録する．登録されたポインタに対し，conditionの作用する関数がベクトル実行されることになる．whileの場合も同じようにαの効果をwhileの作用する関数に及ぼすことができる．

次実行あるいは異なる処理を施すべき対象を縦方向に登録することによって，$ff_V(ff_*(PRI))$のベクトル実行を可能とすることができる．

最後に，$ff_V(ff_V(PRI))$の場合のベクトル化について述べる．αは，前処理としてPRIの作用する対象をHP arrayに登録する動作を行なう．このとき，収集すべきポインタは列となっているから，ポインタの収拾収拾は記憶域への連続アクセスとなり，ベクトル実行の効果が高まる．

αが多重にかかっている場合も同様の動作をHP arrayに登録されているポインタに対して行なえばよい．

これにより，priのベクトル実行が可能となる．

ポインタの収集動作は，HP arrayに登録されているポインタ数と収集されるべき列のポインタ数でまわされる二重DOループで記述される．

一般には収集されるべき列のポインタ数は列毎に異なるので，HP arrayに登録されたポインタ数で回されるDOループをベクトル化する．しかし，αの多重度が大きい場合，列の平均長をL，αの処理回数をnとすると，処理を重ねる毎にHP arrayに登録されるポインタの数は増加する$(\sim L^n)$のに対し，収集されるべき列のポインタ数は常に列の平均長$(\sim L)$にしかならない．

従って，等構造性が示されている場合には，二重ループを入れ換えHP arrayに登録されているポインタ数をベクトル化するようにしてループ長を増加させるようにする．

3. FPのFORTRAN変換過程

現在試作中のFPコンパイラは，FPプログラムをスーパコンピュータ:SX2-N用のFORTRAN(FORTRAN77/SX)プログラムへ自動変換する．そのプログラムを更にSX2-N FORTRANコンパイラがマシン語へ変換するという2段階の変換を経てプログラムは実行される．

4． ベクトル化の効果

以上で述べたFPプログラムの自動変換法の効果を測定するために，変換形式に従って，FPで書かれた行列の積を求めるプログラムをハンドコンパイルし，東北大学大型計算機センターのSX-2N上で実行させた．

下記のFPプログラム上で*で表記してある部分は，対象の等構造性を生かしてプログラミングしてある．また，内積を求める部分では，ベクトル実行に適した原始関数VMULTを使用した．

・行列の積を求めるプログラム

$\alpha*(\alpha(IP))\cdot(\alpha*(dist1))\cdot distr[1,TRANS\cdot 2]$

$IP\equiv(/+)\cdot VMULT$

この計算では，50*50の要素数を持つ正方行列を作用対象としている．実行の結果，処理に要したCPU時間は

VECTOR mode (Tv) : 83.0(ms)
NOVECTOR mode(Ts): 238.6(ms)
ベクトル化率 : 93 %
速度比(Tv/Ts) :2.875

結果より，並列性を持つFPの関数を用いれば我々の提案した変換法により高いベクトル化を持つプログラムが生成されることが分かった．

ベクトル化率の高いわりにスカラ実行との速度比が小さいことが今後の課題と思われる．

5． まとめ

本稿では，並列処理を自然に表現できる言語として関数型言語FPをとらえ，FPをベクトル処理向きに自動コンパイルする手法を述べた．ベクトル処理を生かすために，FPの関数の組み合わせを考え，各々に対して効果的にベクトル処理できるようなFORTRANプログラムを生成するようにした．そのままではベクトル化の効果が出せない場合でも，プログラマが対象の等構造性を指示することによって，より効果的なFORTRANプログラムを生成するようにした．また，行列の積を求めるプログラムを実行させて，変換手法の効果を測定した．

今後の課題としては，再帰関数にも対応するように

すること，ベクトル処理に適した原始関数，プログラム構成子を考えていくことなどがあげられる．

参考文献

[1]平井健治，島崎眞昭，津田孝夫，見城司：ベクトル処理機能を利用したFP処理系VFPシステム，日本ソフトウェア科学会第三会大会論文集 (1986)，PP109-112

[2]辰口和保，村岡洋一：スーパコンピュータ上の並列論理型言語処理系，情報処理学会プログラミング言語研究会，PL-14-3 (1987)

[3]金田泰，菅谷正弘：プログラム変換にもとづくリストのベクトル処理方法とそのエイト・クウィーン問題への適用，情報処理学会論文誌，Vol.30，No.7，pp856-868 (1989)

[4]阿部一裕，安井裕：スーパコンピュータのためのベクトル化Ｌｉｓｐコンパイラ，情報処理学会記号処理研究会，SYM-54-2 (1990)

[5]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)

[6]J.Backus：Can programming be liberated from the Von Neuman style? A functional style and algebra of programs，CACM，Vol.21，No.8，pp.613-641 (1979)

[7]D.Padua，M.Wolfe：Advanced compiler optimizations for supercomputers，CACM，Vol.29，No.29，pp.1184-1200 (1986)

[8]島崎眞昭：スーパーコンピュータとプログラミング，計算機科学/ソフトウェア技術講座９，共立出版

SIGPL 25-2

"The Compiling Method of The Functional Language FP for Vector Processing"

by H. Takemiya (Hitachi Tohoku Software, 2-4-1 Ichibanty o, Aobaku, Sendaishi. 980, Japan), H. Nunokawa, N. Shiratori, S. Noguchi (Reserch Institute of Electrical Communicati on, Tohoku University)

The functional language FP is powerful for parallel processing. To carry out vector processing of FP program, we propose to translate the FP program into FORTRAN . This method can generate highly vectorized FORTRAN programs for all combinations of vector processable FP functions. As a result, both programmer and computer can recognize vector processable parts in the program without analyzing data dependence relationship. We executed matrix multiplication program in FP on the supercomputer SX-2N. It excuted nearly 3 times faster than in the sequential mode and its vectorization ratio is 93 percent.