関数型言語FPの ベクトルプロセッサ向きコンパイル手法

6J－7

武宮 博† 布川 博士‡ 白鳥 則郎‡ 野口 正一‡

†日立東北ソフトウェア ‡東北大学電気通信研究所

1 はじめに

並列処理（パイプライン処理）機能による高速演算処理の可能なベクトルプロセッサが実装している言語はFORTRANのみであるが，FORTRANのDO loopでは並列性を簡潔に表現できない．関数型言語FPは，apply to all(α)を始めとして並列性を簡潔に表現できる関数が存在しており，並列処理向き言語であるといえる．我々は現在スーパーコンピューター（SX-2N）上でFPプログラムを高速に実行することを目的として，FPプログラムをFORRANプログラムに変換するFPコンパイラを制作中である．

2 変換手法

2．1 ＦＰプログラムの高速実行

関数型言語FP[2]は，データである列，原始関数，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，関数を定義し名前をつける定義から構成される（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）．ＦＰプログラムをベクトルプロセッサ上で高速に動かすためには，パイプライン機構に適したデータ構造，原始関数及び関数形式の並列性を活かした変換が必要である．また，最適化も，更にベクトル化率をあげるために必要となる．以下で，それらの手法について述べる．

2．2 FPプログラムの変換過程

FPのプログラムは図１のように二段階のコンパイル過程を経て，ＳＸ－２Ｎ上で実行される．

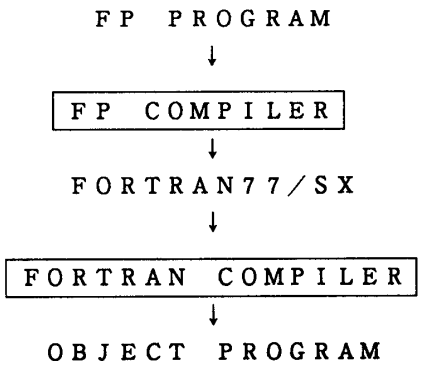


図１

このように，FPプログラムをコンパイルする事によって，インタープリタによる処理より，一層高速化が可能となる．また，二つのコンパイラにより二段階の最適化が可能である．

2．2 データ構造

列処理をパイプライン機構を通して有効に行なうため，[1]と同様に，次のようなデータ構造を採用する（図２）．

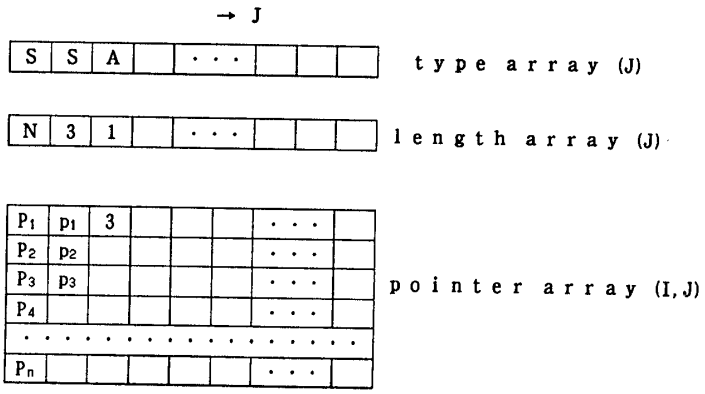


図２

データタイプは 列，整数，真偽値の三つを考え，type arrayにタグとして格納する．length arrayには下部構造のセルの位置を示すポインタの個数を格納する．ポインタarrayには，ポインタをＩ方向に格納する．atomの場合はその値を格納する．ポインタの値は，Ｊ方向の配列のアドレスを示す．この構造は，処理対象となるポインタを連続的にパイプラインに投入できるため，ベクトル処理向きの構造である．

2．3 関数の変換形式

FPの関数には原始関数と関数形式の二種類が存在する．原始関数は，定義[2]に従った処理を行なうFORTRANプログラムに変換する．関数形式の場合には，変換はプログラム構成子の引数である関数の処理部分をはさんで前処理部と後処理部に二分される．例として，α(+)の処理を図３に示す．

Compiling Method of FP Programs for Vector Processor

Hiroshi TAKEMIYA†, Hiroshi NUNOKAWA‡, Norio SHIRATORI‡, Shoichi NOGUCHI‡

Hitachi Tohoku Software Co., Ltd

Research Institute of Electrical Commumication, Tohoku University

```
    HPTEMP=HP(1)
 *VDIR NODEP
    DO I=1,LENGTH(HPTEMP)
     HP(I)=INTER(I,HPTEMP)
    END DO
 *VDIR NODEP
    DO I=1,LENGTH(HPTEMP)
     INTER(1,HP(I))=INTER(1,INTER(1,HP(I)))
                    +INTER(1,INTER(2,HP(I)))
     NTYPE(HP(I))='atom'
         LENGTH(HP(I))=1
    END DO
   HP(1)=HPTEMP
```

図 3

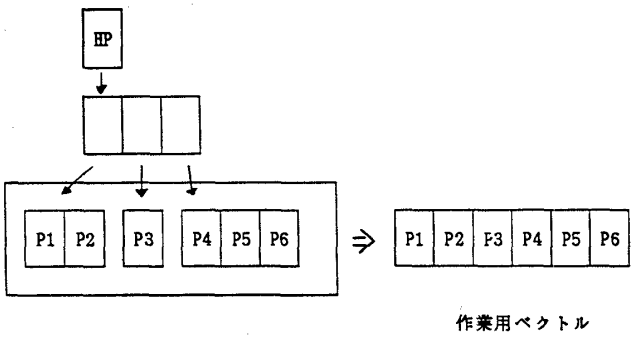
αによる処理をベクトル化するため，上位セルの位置を示すHPは配列として与えられる．通常の場合，最上位セルの位置を示すポインタがHP(1)に格納されるが，α起動時は前処理としてHP(1)をHPTEMPに待避させ，深さが1番目のセルのポインタをHP(1)に格納する．その後，+の操作を行なった後，後処理としてHP(1)に最上位セルの位置を示すポインタを格納する．

2．4　最適化

FPプログラムをベクトルプロセッサ上で高速に実行するためには，コンパイル時にできる限りベクトル実行可能な部分を抽出しなければならない．そのためには，関数の定義にしたがって処理するとベクトル実行不可能なものを，関数の意味を考えることによって，ベクトル実行可能にする必要がある．

たとえば，α がn重にnestしている場合，定義通り行なうとn重にnestしたDo loopとなるが，各列の長さが一般に異なるため最内側loopを除いて逐次実行となってしまい，αの並列性を十分に生かすことができない．そこで，FORTRAN変換時には，ポインタの収集を行ない，深さが1番目の列を結合させて作業用ベクトルを作ることによってベクトル実行可能部分を増加させる．[1]（図4）

この処理は二重Do loopを用いて行なわれるが，処理を重ねる毎に長くなる作業用ベクトルのloopが外側ループになるため，作業用ベクトルを作る処理時のベクトル長は常に列の平均長程度（～Lとする）にしかならない．一方作業用ベクトルのベクトル長は，n回の作業後，～L^{n}程度になっている．よって，二重Do loopを入れ換え，マスクベクトルを利用してポインタの収集を行なう．この方法により，作業用ベクトルのベクトル長を活かすことができる．



＜図4＞

このような最適化機能を強化することにより，一層高速化が強化されると考えられる．

3．内積プログラムの実行

ベクトルプロセッサによる高速実行の効果を測定するために，変換形式にしたがって，FPでかかれた内積プログラム[2]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，ACOS-2020上で実行させた．この計算では，要素数が100のベクトルをデータとして使用している．実行の結果，処理に要したcpu時間は，ACOS2020：0.469(ms)，SX-2N(CP)：0.309(ms)，SX-2N(AP)：0.066(ms)となった．

以上の結果より，スカラー実行に比べてベクトル実行は5倍程度処理時間が短縮されており，ベクトル化の効果が出ているといえる．

4．まとめ

本稿では，現在制作中であるFPコンパイラの目的及び変換手法の概要を示した．また，FPで書かれた内積プログラムをハンドコンパイルする事によってSX-2N上で実行させ，ベクトル化の効果を示した．

参考文献

[1]平井健治，島崎真昭，津田孝夫，見城司：ﾍﾞ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回大会論文集(1986)，pp109-112

[2] 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)