ストリームFIFO方式に基づくベクトル・プロセッサ『順風』 －IF文を含むDOループの処理－

7L－4

弘中哲夫　岡崎恵三　久我守弘　村上和彰　富田眞治

(九州大学)

1. はじめに

我々は，従来のベクトル演算方式に比べて，より柔軟なベクトル処理およびベクトル－スカラ協調処理を可能とする“ストリームFIFO方式”[1]を提案し，その試作プロセッサ『順風』の開発を行っている[2]．

本稿では，一般にベクトル化が困難だとされている「IF文を含むDOループ」の『順風』における処理方法について述べる．

2. ストリームFIFO方式および『順風』の概要

2. 1 ストリームFIFO方式

ストリームFIFO方式は，以下の特長を持つ[1][2]．

①FIFOレジスタ：FIFO動作するベクトル・レジスタであり，一時に処理可能なベクトル長を制限しない．よって，ストリップ・マイニング処理が不要となる．

②ベクトル－スカラ協調処理：FIFOレジスタは，ベクトル命令およびスカラ命令の双方からアクセス可能である．これにより，スカラ命令のループでもFIFOレジスタ内のベクトル・データに対する演算を行える．

③局所データフロー制御：実行中の命令（ベクトルおよびスカラ命令）間のデータ依存関係に従って，各種資源（ロード／ストア・パイプライン，演算パイプライン，スカラユニット，FIFOレジスタ）をチェイニングする．

2. 2『順風』

『順風』は，ストリームFIFO方式に基づくベクトル・プロセッサの1具現化例である．2. 1節で述べた方式上の特長に加え，ハードウェア構成として「パイプライン共有MIMD」の概念を取り入れている．

すなわち，1本のパイプラインを複数の命令で時分割共有する．個々の命令から見たパイプラインを“仮想パイプライン”と，また，実在するパイプラインを“実パイプライン”と呼ぶ．これにより，次の効果が期待される．

①実パイプラインの使用率向上：1個のベクトル命令に実パイプラインを独占的に使用させた場合，データ依存関係などによるソースオペランド待ちが原因でパイプラインに遊びが生じる．このとき本方式では，命令を切り換えることで遊びの影響を軽減する．

②一時に実行対象とできる命令数の向上：命令のディスパッチ対象を実パイプラインでなく仮想パイプラインとすることで，より多くの命令の同時実行を可能とする．

3. 「IF文を含むDOループ」の処理

3. 1 従来の対処方法

「IF文を含むDOループ」のベクトル化は，一般に困難である．従来のベクトル・プロセッサおよび自動ベクトル化コンパイラは，以下の方法で対処している．

①完全ベクトル化可能なループ：IF文のTHEN項およびELSE項に対応するマスクベクトルを用いる．マスクの使用方法には，次の3通りがある．

a) マスク付きベクトル演算
b) ベクトル収集／拡散
c) リストベクトル・アクセス

②部分ベクトル化可能なループ：ループをベクトル化可能なループとベクトル化不可能なループに分割する．ベクトル化可能なループは①の方法で対処し，ベクトル化不可能なループはスカラ命令のループとする．

③ベクトル化不可能なループ：ループ外への飛び出しがある場合などに相当し，スカラ命令のループのままである．

3. 2 問題点

3.1節で述べた方法には，以下の問題点がある．

①完全ベクトル化可能なループ：マスクベクトルの真率（マスクが1である比率）により，方法a, b, cのいずれを選択するかが問題となる．一般には，真率が高い場合は方法aを，低い場合には方法b, cを用いる．しかし，IF文の条件次第ではコンパイル時に真率を正確に判定するのは困難である．この場合は，複数の方法に基づくオブジェクトコードを生成し，実行時にいずれかを選択することで対処する．

②部分ベクトル化可能なループ：ベクトル化可能なループとベクトル化不可能なループとの間にデータ依存関係が存在することがある．このとき，ベクトル・ユニットとスカラ・ユニットとの間でデータの授受が必要となる．このデータ授受の方法には，メモリ経由とレジスタ経由の2つの方法が可能だが，メモリ経由は遅い，レジスタ経由で行う場合，従来のベクトル・レジスタでは通常のスカラ命令から直接アクセス不可能なので，ベクトル・レジスタ－スカラ・レジスタ間データ転送を別に行わなければならない．しかも，同一ベクトル・レジスタに対して，通常のベクトル命令によるアクセスと当該データ転送とをデータ依存関係を保証しながら同時に行うのは，困難である．よって，これら2つのレジスタ・アクセスは一般にはオーバラップされない．

4. 『順風』における「IF文を含むDOループ」の処理

4. 1 方針

3. 2節で述べた問題点に対して，『順風』では以下の対処を行う．

【dʒúmpuː】: A Vector Processor Based on Stream FIFO Architecture : On Do Loops Including IF Statements

Tetsuo HIRONAKA, Keizou OKAZAKI, Morihiro KUGA, Kazuaki MURAKAMI, and Shinji TOMITA

①完全ベクトル化可能なループ：マスクベクトルの真率に依存しないように，ベクトル収集／拡散を拡張した“ベクトル分配／併合”処理方式を採用する．さらに，IF文のTHEN項とELSE項を同時処理可能とする．
②部分ベクトル化可能なループ：ベクトル・ユニットとスカラ・ユニット間のデータ授受にFIFOレジスタを用いる．これにより，ベクトル命令およびスカラ命令からの同一レジスタへの同時アクセスを可能とする．

4. 2 関連命令

以下のベクトル命令を定義する．なお，以下で単にレジスタと言った場合，FIFOレジスタあるいはスカラレジスタを指す．
①ベクトル比較命令（vCOMP）：2個のソースレジスタに対して比較演算を行う．比較結果をマスクベクトルとして，デスティネーション（複数個指定可能）であるマスクFIFOレジスタに格納する．
②マスク付きベクトル・ロード／ストア命令（vLOADm, vSTOREm）：メモリ上のベクトルデータにおいて，マスク値1に対応するアドレスから／へ，ベクトル要素をロード／ストアする．
③ベクトル分配命令（vSWITCHm）：マスクベクトルに従って，ソースレジスタから読み込んだデータを2個のデスティネーションレジスタに分配する．分配し終ると，その旨を示すコンディションコード（End__Of__Stream）を両デスティネーションレジスタに送る．
④ベクトル併合命令（vMERGEm）：vSWITCHm命令と反対の動作を行う．すなわち，マスクベクトルに従って，2個のソースレジスタのいずれか一方からデータを読出し，1個のデスティネーションレジスタに格納する．

4. 3 処理過程

図1の例を用いて，完全ベクトル化可能な「IF文を含むDOループ」に対する“ベクトル分配／併合”処理を説明する．
(1) まず，ⓐのvLOAD命令でベクトルAをロードする．ここで，ベクトルAはIF文の条件判定とループ本体の演算の双方に用いるので，2個のFIFOレジスタF0, F1両方にロードする．
(2) ⓑのvCOMP命令は，F0のベクトルAに対して比較演算（IF文の条件判定）を行う．その結果はマスクベクトルとして，マスクFIFOレジスタM0～M3に格納する．
(3) ⓒのvSWITCHm命令は，M0のマスクベクトルに従って，F1のベクトルAをFIFOレジスタF2（THEN項）とF3（ELSE項）に分配する．これ以降は，THEN項とELSE項とで並列に実行が進む．ベクトルAの分配が終了すると，End__Of__StreamをF2とF3に送る．
(4) まず，THEN項では，ⓓのvLOADm命令がM1のマスクベクトルに従ってベクトルBをマスク付きロードし，ⓔのvSUB命令がベクトルAとBの減算を行う．一方，ELSE項では，ⓕのvLOADm命令がM2のマスクベクトルに従ってベクトルCをマスク付きロードし，ⓖのvADD命令がベクトルAとCの加算を行う．以上の演算は，F2ないしF3にEnd__Of__Streamが検出されるまで行われる．
(5) ⓗのvMERGEm命令は，M3のマスクベクトルに従って，F6（THEN項の減算結果）とF7（ELSE項の加算結果）のいずれか一方のデータを読出し，それをF8に格納する．すなわち，THEN項とELSE項を併合する．
(6) 最後に，①のvSTORE命令がF8をベクトルDとしてメモリにストアする．

5. おわりに

以上，『順風』における「IF文を含むDOループ」の処理について述べた．現在，『順風』シミュレータを用いて，処理方式およびハードウェア構成方式の検証，動作解析，性能評価等を進めている．本稿で提案した“ベクトル分配／併合”処理方式の有効性については，別の機会に報告したい．

参考文献

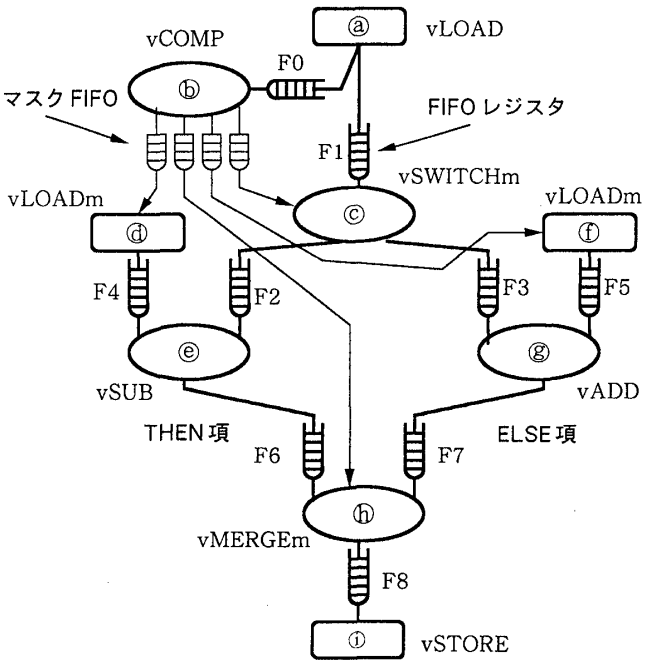
[1] 弘中ほか：ストリームFIFO方式の構想－SIMP方式のベクトル処理への適応－，情処38全大論文集，5T-10（1989年3月）．
[2] 弘中ほか：ストリームFIFO方式に基づくベクトル・プロセッサ『順風』，信学技法，CPSY89-39（1989年8月）．

```
      DO 10 I=1,1000
        IF A(I).GT.0. THEN
          D(I)=A(I)-B(I)
        ELSE
          D(I)=A(I)+C(I)
        ENDIF
10    CONTINUE
```

(a) ソースコード

```
vLOAD     F0, F1 ← A (I)                ⓐ
vCOMP     M0, M1, M2, M3, ← F0 .GT. 0   ⓑ
vSWITCHm  F2,F3 ← F1 by M0              ⓒ
vLOADm    F4 ← B (I) by M1              ⓓ
vSUB      F6 ← F2, F4                   ⓔ
vLOADm    F5 ← C (I) by ! M2            ⓕ
vADD      F7 ← F3, F5                   ⓖ
vMERGEm   F8 ← F6, F7, by M3            ⓗ
vSTORE    D (I) ← F8                    ①
```

(b) オブジェクトコード



(C) 演算実行の流れ

図1 IF文を含むDOループの例