計算機アーキテクチャ 85－3
（1990．11．21）

『順風』：ストリームFIFO方式に基づくシングルチップ・ベクトルプロセッサ・プロトタイプ －「IF 文を含むDOループ」への対処法－

岡崎恵三　弘中哲夫　村上和彰　富田眞治
（九州大学大学院総合理工学研究科）

シングルチップ・ベクトルプロセッサのプロトタイプ機『順風』を開発している．ベクトル演算方式として，我々が提案する“ストリームFIFO方式”を採用する．さらに，パイプラインの稼働率を向上させるため「パイプライン共有MIMD」の概念に基づく“仮想パイプライン”を採用している．

本稿では，通常ベクトル化が困難と言われている「IF文を含むDOループ」に対する『順風』の対処法である，①ベクトル分配／併合，②ベクトル／スカラ協調処理，③ベクトル命令実行停止，について述べる．さらに，マスク付きベクトル演算方式，ベクトル分配／併合方式，リストベクトル・アクセス方式に関する『順風』の性能をシミュレーションより評価する．

【d ʒ úmp u :】: A Single-Chip Vector-Processor Prototype Based on Streaming／FIFO Architecture
－Handling “DO-Loops Including IF Statements”－
(in Japanese)

Keizou OKAZAKI, Tetsuo HIRONAKA, Kazuaki MURAKAMI, and Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyusyu University
6-1, Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
e-mail : okazaki@kyu-is.is.kyushu-u.ac.jp

A prototype of a single-chip vector processor 【d ʒ úmp u :】 is under development at Kyushu University. 【d ʒ úmp u :】 is based on “Streaming／FIFO architecture”, which adopt FIFO-register as operands of vector and scalar instructions. Moreover, arithmetic and load／store pipelines are virtualized in the manner of “Pipelined MIMD”.

This paper presents several method to handle “DO-loops including IF statements”, such as 1) vector switch／merge, 2) vector-scalar cooperation, 3) termination of vector execution. Simulation results for vector switch／merge execution are also presented.

1 はじめに

我々は，従来のベクトル演算方式に比べ，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]を提案している．本方式はまた，将来の1000万トランジスタ時代をにらんで，ベクトル・プロセッサのシングルチップ化を目指したアーキテクチャである．すなわち，従来のベクトル・プロセッサのアーキテクチャは極めて高いメモリバンド巾に依存しているが，シングルチップ・ベクトルプロセッサにおいてはピン数の制限によりこのような高メモリバンド巾を前提としたアーキテクチャは不適であると考える．つまり，ストリームFIFO方式は，オフチップ通信量に対する性能比の向上を目的としている．ストリームFIFO方式の有効性を実証し，かつ，問題点を探るために，プロトタイプ・プロセッサ『順風』の開発を行っている[2]．

ベクトルプロセッサにおいて高い実効性能を得るには，可能な限りベクトル化率を高くする必要がある．したがって，実際のプログラムによく含まれベクトル化率低下の要因[7]となる「IF文を含むDOループ」および「疎行列に対する演算」への対処法が課題となる．

本稿では，「IF文を含むDOループ」に対する『順風』の処理法である，①ベクトル分配／併合，②ベクトル／スカラ協調処理；③ベクトル命令実行停止，について述べる．また，マスク付きベクトル演算方式，ベクトル分配／併合方式，リストベクトル・アクセス方式に関する『順風』の性能をシミュレーションより評価する．

2 概要

2. 1 ストリームFIFO方式

ストリームFIFO方式は以下の特徴を持つ．

① ベクトル・レジスタとしてFIFOレジスタを用いることにより，一時に処理可能なベクトル長を制限しない．すなわち，ストリップ・マイニング処理が不要となり，メモリ・アクセス量を削減する．

② スカラ命令およびベクトル命令の双方からFIFOレジスタにアクセスできる．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が行える．すなわち，小さなオーバヘッドで“ベクトル—スカラ協調処理”が可能となる．

③ 実行中の命令（ベクトルおよびスカラ）間のデータ依存関係に従って，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レジスタを動的にチェイニングし，複数命令の同時実行を行う．

2. 2 仮想パイプライン

『順風』では，このストリームFIFO方式をハードウェアで実現する上で，さらに「パイプライン共有MIMD」の概念を演算パイプラインおよびロード／ストア・パイプラインに取り入れている[4]．すなわち，図1で示すように1個の命令が独占的に1本のパイプラインを使用するのでなく，1本のパイプラインを複数の命令で時分割共有する．この時，個々の命令から見たパイプラインを“仮想パイプライン”と呼び，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できる．

① 実パイプラインの使用率の向上：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．この時，本方式ではパイプラインを使用する命令を切り換えることで遊びの影響を軽減する．

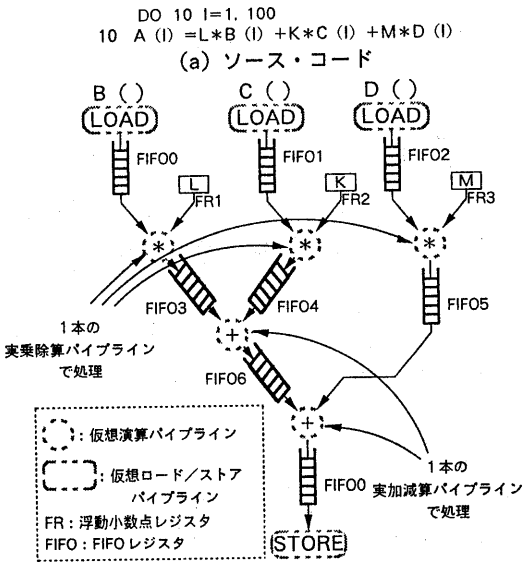

図1. 仮想パイプラインの概念

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．

2. 3 『順風』の諸元

『順風』ベクトルユニットの諸元を以下に示す．

(1) FIFOレジスタ：データ64ビット，コンディションコード5ビットの合計69ビット巾で深さは32要素である．FIFOレジスタは浮動小数点データおよび整数データを格納するのに使用される．FIFOレジスタは16本存在する．

(2) マスクFIFOレジスタ：マスク値1ビット，コンディションコード1ビット，演算指定コード6ビットの合計8ビット幅で深さは32要素である．マスクFIFOレジスタは16本存在する．

(3) 仮想パイプライン数：仮想演算パイプライン8本，仮想ロード／ストアパイプライン8本．

(4) 実パイプライン数：加減算パイプライン1本，乗除算パイプライン1本，実ロード／ストアパイプライン2本．

3 「IF文を含むDOループ」への対処

3. 1 従来の対処法

「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

ベクトル比較演算命令等で，IF文のTHEN項あるいはELSE項に対応するマスクベクトルを生成する．マスクベクトルを用いたベクトル化方法には，次の2方式がある[6][7]．

① マスク付きベクトル演算方式：マスク付きベクトル演算命令が，通常のベクトル演算命令同様すべてのベクトル要素に対して演算を行う．ただし，対応するマスクベク

トル要素が真（あるいは偽）であるベクトル要素に対する演算結果のみを格納する.

② ベクトル収集／拡散（gather／scatter）方式：まず，対応するマスクベクトル要素が真（あるいは偽）であるベクトル要素のみを収集したベクトルを生成する．演算は，収集したベクトルに対して行う．したがって，①のように元のベクトルの全要素に対して演算を行う必要がない．演算結果であるベクトルに対しては，各要素がベクトル内の正しい位置に格納されるようにこれを拡散する．ベクトル収集／拡散方式は，さらに次の2方式に分類される．

a) ベクトル圧縮／伸長（compress／expand）方式：収集／拡散すべきベクトル要素のインデックスが，対応するマスクベクトル要素のインデックスで直接与えられる．ベクトル圧縮／伸長命令により，ベクトルレジスタ間あるいはベクトルレジスタ－メモリ間で，ベクトルの収集／拡散を行う．

b) リストベクトル・アクセス方式：収集／拡散すべきベクトル要素のインデックスが，インデックス・リストにより与えられる．これにはまず，マスクベクトルからインデックス・リストを作成して，元のベクトルがリスト・ベクトルとしてアクセス出来るようにする必要がある．そして，リストベクトル・アクセス命令によりリスト・ベクトルをロード／ストアすることで，当該ベクトルの収集／拡散を行う．

(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部分ループとベクトル化不可能な部分ループに分割する．ベクトル化可能な部分ループは(1)の方法で対処し，ベクトル化不可能な部分ループはスカラ命令のループのままとする．

(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等に相当し，スカラ命令のループのままである．

3. 2 問題点

前節で述べた対処方法には，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により，マスク付きベクトル演算方式，ベクトル圧縮／伸長方式，リストベクトル・アクセス方式のいずれのを選択するかが問題になる．直感的には，真率が高い場合はマスク付きベクトル演算方式，低い場合はベクトル圧縮／伸長方式またはリストベクトル・アクセス方式を用いるのが良さそうである．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．さらに，真率だけでなく，THEN項およびELSE項それぞれのベクトル命令数も性能を左右する要因となるので，簡単には決められない．

(2) 部分ベクトル化可能なループ

ベクトル化可能な部分ループとベクトル化不可能な部分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－スカラレジスタ転送を別に行う必要がある．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オーバラップされない．

3. 3 『順風』の対処方針

『順風』では，以下の方針にて「IF文を含むDOループ」に対処する．

(1) 完全ベクトル化可能なループ

ベクトル収集／拡散方式の1方式として，ストリームFIFO方式と親和性のよい“ベクトル分配／併合（switch／merge）”方式を導入する[3]．ベクトル分配／併合方式は，THEN項とELSE項が同一のソースおよびデスティネーションを有する場合に特化したものである．まず，対応するマスクベクトル要素の真偽に従って，元の各ベクトル要素を2個のベクトル（真ベクトルおよび偽ベクトル）のいずれかに分配する．THEN項およびELSE項の演算は，それぞれ真ベクトルおよび偽ベクトルに対して行う．これは，同時に実行可能である．演算結果である2個のベクトルは，同一マスクベクトル要素の真偽に従って1個のベクトルに併合される．

(2) 部分ベクトル化可能なループ

表1. 各種ベクトル化方式における所要処理量の比較

		ロード回数	演算回数	ストア回数	オーバヘッド
マスク付ベクトル演算	マスク付ストアなし	$l_T+l_E+l_C+s_T+s_E$	$t+e$	$s_T+s_E+s_C$	
	マスク付ストア有り	$l_T+l_E+l_C$	$t+e$	$s_T+s_E+s_C$	
ベクトル圧縮／伸長	伸長ストアなし	$l_T+l_E+l_C$	$t\times f+e\times(1-f)$	$s_T+s_E+s_C$	圧縮：$l_T+l_E+2\times l_C$ 伸長：$s_T+s_E+2\times s_C$
	伸長ストア有り	$l_T+l_E+l_C$	$t\times f+e\times(1-f)$	$s_T+s_E+2\times s_C$	圧縮：$l_T+l_E+2\times l_C$
リストベクトル・アクセス		$l_T\times f+l_E\times(1-f)+l_C$	$t\times f+e\times(1-f)$	$s_T\times f+s_E\times(1-f)+s_C$	リスト生成：2
ベクトル分配／併合	伸長ストア有り	$l_T+l_E+l_C$	$t\times f+e\times(1-f)$	$s_T+s_E+s_C$	分配：l_C 併合：s_C

l_T：THEN項のみでロードするベクトル数
l_E：ELSE項のみでロードするベクトル数
l_C：THEN項とELSE項とで共通にロードするベクトル数
t：THEN項のベクトル命令数
e：ELSE項のベクトル命令数
s_T：THEN項のみでストアするベクトル数
s_E：ELSE項のみでストアするベクトル数
s_C：THEN項とELSE項とで共通にストアするベクトル数
f：真率（$0\leq f\leq 1$）

“ベクトルースカラ協調処理”により，ベクトル化可能な部分ループとベクトル化不可能な部分ループとの間のデータ依存関係に効率よく対処する．すなわち，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，さらには，部分ループ同士のオーバラップ実行を可能とする．

(3) 完全ベクトル化不可能なループ

ループ外への飛び出しのあるループは，ベクトル命令の途中実行停止を必要とすることから，従来のベクトル・プロセッサではベクトル化不可能であった．『順風』では“ベクトル命令実行停止”機能を備えることで，ループ外への飛び出しのあるループのベクトル化を可能とする．

3. 4 ベクトル化方式の比較

『順風』では，完全ベクトル化可能なループのベクトル化方式として，

- マスク付きベクトル演算方式
- ベクトル圧縮／伸長方式
- ベクトル分配／併合方式
- リストベクトル・アクセス方式

の4方式を採用する．表1に，これら4方式を用いた場合にループ実行に要する処理量を示す．ただし，ループは1個のIF－THEN－ELSEのみから構成されているものとする．また，所要処理には，マスクベクトル生成処理を含まない．

4 『順風』における「IF文を含むDOループ」の処理

4. 1 関連命令

まず，関連する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) 一般命令

①ベクトル演算命令（VADD, VMUL等）：2個のソースレジスタに対してベクトル演算を行い，最大2個のデスティネーションレジスタに結果を格納する．

②ベクトルロード命令（VLOAD）：メモリから最大2個のデスティネーションレジスタにベクトル（連続または等間隔ベクトル）をロードする．

③ベクトルストア命令（VSTORE）：ソースレジスタからメモリにベクトル（連続または等間隔ベクトル）をストアする．

④ベクトルコピー命令（VCOPY）：1個のソースレジスタの値を2個のデスティネーションレジスタにコピーする．

⑤ベクトル比較命令（VCOMP）：2個のソースレジスタに対してベクトル比較演算を行う．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演算関連命令

①マスク付きベクトル演算命令（VADDM, VMULM等）：対応するマスクベクトル要素が真であるベクトル要素に対してのみ演算を行い，デスティネーションレジスタに演算結果を格納する．従来のベクトルレジスタにおけるマスク付きベクトル演算命令の定義と異なり，対応するマスクベクトル要素が偽であるベクトル要素に対しては，空データを生成・格納する．

②マスク付きベクトルロード命令（VLOADM）：メモリ上のベクトルに対して，対応するマスクベクトル要素が真であるベクトル要素のみをデスティネーションレジスタにロードする．対応するマスクベクトル要素が偽であるベクトル要素に対しては，空データをロードする．

③マスク付きベクトルストア命令（VSTOREM）：ソースレジスタ上のベクトルに対して，対応するマスクベクトル要素が真であるベクトル要素のみをメモリにストアする．対応するマスクベクトル要素が偽であるベクトル要素に対しては，ストアを行わない．

(3) ベクトル圧縮／伸長関連命令

①ベクトル圧縮ロード命令（VCLOADM）：メモリ上のベクトルに対して，対応するマスクベクトル要素が真であるベクトル要素のみをデスティネーションレジスタにロードする．対応するマスクベクトル要素が偽であるベクトル要素に対しては，ロードを行わない．

②ベクトル伸長ストア命令（VESTOREM）：メモリ上のベクトルに対して，対応するマスクベクトル要素が真であるベクトル要素へソースレジスタからストアする．対応するマスクベクトル要素が偽であるベクトル要素へは，ストアしない．

(4) ベクトル分配／併合関連命令

①ベクトル分配命令（VSWITCHM）：対応するマスクベクトル要素の真偽に従って，ソースレジスタから読み込んだベクトル要素を2個のデスティネーションレジスタに分配する．分配し終わると，その旨を示すコンディションコード（EOS：End__Of__Stream）を両デスティネーションレジスタに送る．EOSには，次の2種類がある．

a) EOS__V：ソースレジスタからの最終データに付随するEOSで，データが有効である旨を示す．

b) EOS__I：ソースレジスタからの最終データを受け取らない側のレジスタには，分配終了を通知するための空データを分配する．この空データに付随するEOSで，データが無効である旨を示す．

②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，対応するマスクベクトル要素の真偽に従って，2個のソースレジスタのいずれか一方からベクトル要素を取り出し，1個のデスティネーションレジスタに格納する．

(5) リストベクトル・アクセス関連命令

①インデックスリスト生成命令（VGENASM）：初期値0の等差数列を生成する．公差は，ソーススカラレジスタにより与える．生成される数列のデスティネーションFIFOレジスタへの格納は，マスクベクトルにより制御される．数列の最終データには，EOS__VまたはEOS__Iが付加される．

②リストベクトル・ロード／ストア命令（VLOADIX／VSTOREIX）：リストベクトルをロード／ストアする．インデックスリストは，ソースFIFOレジスタにより与える．

(6) ベクトル命令実行停止関連命令

①ベクトル実行停止命令（VWHILE）：VCOMP命令同様，2個のソースレジスタに対してベクトル比較演算を行う．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．ただし，VCOMP命令と異なり，比較結果が一度でも偽となったらEOSをすべてのデスティネーションに送り，実行を完了する．これに伴い，ベクトル実行停止モードで実行中のすべてのベクトル命令の実行が停止される．

②ベクトル実行停止モード：VWHILE命令の終了とともに，実行が停止されるモード．VWHILE命令以外のいずれのベクトル命令も，本モード下で実行され得る．命令ニーモニックの接尾に「．w」を付加することで，本モード下で実行するベクトル命令を表記する．

```
      DO 10 I=1,100
        IF (A (I) .GT. 0.0) THEN
           D (I) =A (I) −B (I)
        ELSE
           D (I) =X∗A (I) +C (I)
        ENDIF
10    CONTINUE
```

(a) ソース・プログラム
(テストプログラム1)

```
MOVE      VLR←#100 ·····ⓐ
VLOAD     F0,F1←A () ·····ⓑ
VCOMP     M0,M1,M2,M3←F0.GT.#0.0 ·····ⓒ
VSWITCHM  F2,F3←F1 BY M0 ·····ⓓ
VCLOADM   F4←B () BY M1 ·····ⓔ
VSUB      F6←F2,F4 ·····ⓕ
VCLOADM   F5←C () BY !M2 ·····ⓖ
VMUL      F7←F3,FR2 ·····ⓗ
VADD      F8←F5,F7 ·····ⓘ
VMERGEM   F9←F6,F8 BY M3 ·····ⓙ
VSTORE    D () ←F9 ·····ⓚ
```

(b) オブジェクト・コード

図2. 完全ベクトル化可能なループ

(7) ベクトル長

ベクトル長は，ベクトル長レジスタ（VLR）で与える．ただし，以下のベクトル命令は，指定ベクトル長に達しなくてもEOSにより実行を完了する場合がある．

・ベクトル演算命令，マスク付きベクトル演算命令
・ベクトルコピー命令
・すべてのベクトルロード／ストア命令

いずれかのソースレジスタからEOS_Vの付加したデータあるいはマスクを取り出した場合，演算を行ったのち実行を完了する．一方，EOS_Iの付加したデータあるいはマスクを取り出した場合，演算を行わずに直ちに実行を完了する．いずれの場合も，ソースレジスタから取り出したEOSをそのままデスティネーションレジスタに伝搬する．

4. 2 ベクトル分配／併合

図2の例を用いて，ベクトル分配／併合方式を説明する．なお，レジスタに関して，以下の表記法を用いる．

・F0～15：FIFOレジスタ
・M0～15：マスクFIFOレジスタ
・GR0～31：汎用（スカラ）レジスタ
・FR0～31：浮動小数点（スカラ）レジスタ

① まず，ⓐのMOVE命令でベクトル長を設定する．
② ⓑ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とF1にロードする．
③ ⓒのVCOMP命令は，F0のベクトルAに対して比較演算（IF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．
④ ⓓ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THEN項側のF2（真ベクトルA）とELSE項側のF3（偽ベクトルA）に分配する．これ以降は，THEN項とELSE項とで並列に処理が進む．ベクトルAの分配が終了すると，EOS（EOS_VまたはEOS_I）をF2とF3に送る．

```
      DO 10 I=1,100
         IF (A (I).GT.0.0) THEN
            D (I) =A (I) −B (I)
               DO 20 K=1,3
                  IF (D (I).LE.0.0) THEN
                     GOTO 10
                  ELSE
                     E (K) =D (I) +F (K)
                  ENDIF
20             CONTINUE
10    CONTINUE
```

(a) ソース・プログラム

```
          MOVE      VLR ← #100 ····· ⓐ
          VLOAD     F0,F1 ← A () ····· ⓑ
          VCOMP     M0,M1 ← F0.GT.0.0 ····· ⓒ
          VSWITCHM  F2,FR0 ← F1 by M0····· ⓓ
          VCLOADM   F3 ← B () by M1····· ⓔ
          VSUBM     F5,F4 ← F2,F3 ····· ⓕ
          VESTOREM  D () ← F4····· ⓖ

L1 :      TEST      TF1 ← F5,EOS_I····· ⓗ
          BRANCH    PC ← END,TF1 ····· ⓘ
          TEST      TF2 ← F5,! EOS_V····· ⓙ
          MOVE      FR1 ← F5 ····· ⓚ
          MOVE      GR1 ← 0 ····· ⓛ
L2 :      ADD       GR1 ← GR1,#1 ····· ····· ⓜ
          SUB       FR2 ← FR1,#0.0 ····· ⓝ
          TEST      TF3 ← FR2,LE ····· ····· ⓞ
          BRANCH    PC ← L3,TF3····· ⓟ
          LOAD      FR3 ← F (GR1)····· ⓠ
          ADD       FR4 ← FR1,FR3····· ⓡ
          STORE     E (GR1) ← FR4 ····· ⓢ
          SUB       GR2 ← GR1,#3····· ⓣ
          TEST      TF4 ← GR2,! ZERO ····· ⓤ
          BRANCH    PC ← L2,TF4 ····· ⓥ
L3 :
          BRANCH    PC ← L1,TF2 ····· ⓦ
END :
```

(b) オブジェクト・コード

図3. 部分ベクトル化可能なループ

⑤ THEN項では，ⓔのVCLOADM命令がM1のマスクベクトルに従ってベクトルBを圧縮ロードし，ⓕのVSUB命令が真ベクトルAと圧縮ベクトルBの減算を行う．一方，ELSE項では，ⓖのVCLOADM命令がM2のマスクベクトルに従ってベクトルCを圧縮ロードする．そして，ⓗのVMUL命令が偽ベクトルAとスカラX（FR2）との乗算を行った結果に，ⓘのVADD命令が圧縮ベクトルCを加算する．ベクトル命令ⓕⓗⓘの実行は，それぞれF2，F3，F7からEOSを取り出すまで続く．
⑥ ⓙのVMERGEM命令は，M3のマスクベクトルに従って，F6（THEN項の結果）とF8（ELSE項の結果）のいずれか一方のデータを読み出し，それをF9に格納する．
⑦ 最後に，ⓚのVSTORE命令がF9をベクトルDとしてメモリにストアする．

4. 3 ベクトル－スカラ協調処理

図3の例を用いて，ベクトル－スカラ協調処理を説明する．図

```
DO 10 I=1,100
   A (I) =C (I) +D (I)
   IF (A (I) .GT. 0.0) GOTO 20
   D (I) =A (I) −B (I)
10 CONTINUE
20 CONTINUE
```

(a) ソース・プログラム

```
MOVE      VLR←#100 ·····ⓐ
VLOAD.w   F0←C () ·····ⓑ
VLOAD.w   F1←D () ·····ⓒ
VADD.w    F2,F3←F0,F1 ·····ⓓ
VWHILE    M0,M1,M2←F2.LE.#0.0 ·····ⓔ
VLOADM    F4←B () BY M0 ·····ⓕ
VSUBM     F5←F3,F4 BY M1 ·····ⓖ
VSTOREM   D () ←F5 BY M2 ·····ⓗ
```

(b) オブジェクト・コード

図4. 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DOループ)

```
DO 10 I=1,100
   IF (A (I) .GT. 0.0) THEN
      D (I) =A (I) −B (I)
   ELSE
      D (I) =X*A (I) +Y*C (I)
   ENDIF
10 CONTINUE
```

(a) テストプログラム2

```
DO 10 I=1,100
   IF (A (I) .GT. 0.0) THEN
      D (I) =X*A (I) +C (I)
   ENDIF
10 CONTINUE
```

(b) テストプログラム3

図5. ベンチマーク・プログラム

3 (a) の内側ループはベクトル化不可能であるが, その外側の文はベクトル化可能である. よって, 図3 (b) に示すようにループを分割する. ベクトル化可能な部分ループは, ベクトル圧縮／伸長方式によりベクトル化されている.

命令ⓐ～ⓚがベクトル化可能な部分ループに相当する. 命令ⓐ～ⓕ, ⓚは, それぞれ図2 (b) の命令ⓐ～ⓕ, ⓚに対応する. ただし, ベクトル分配／併合ではなくベクトル圧縮／伸長によりベクトル化されているので, ⓓのVSWITCHM命令は実質的にベクトル圧縮命令のように振舞う. また, ⓚのVESTORE命令は, 圧縮ベクトルDを伸長しながらストアする. なお, ベクトル化不可能な部分ループへ圧縮ベクトルDを渡すため, ⓕのVSUB命令はF5にも演算結果を格納している.

残りのスカラ命令ⓗ～ⓦがベクトル化不可能な部分ループを構成する. 命令ⓗⓘおよび命令ⓘⓦの条件分岐関連命令[6]は, 外側ループの戻り分岐に相当する. また, 命令ⓣⓤⓥが内側ループの戻り分岐に, 命令ⓝⓞⓟがIF文とループ外への飛び出しに, それぞれ相当する.

ベクトル命令ⓑ～ⓚを実行するベクトルユニットとスカラ命令ⓗ～ⓦを実行するスカラユニットは, FIFOレジスタF5に関

```
MOVE      VLR←#100
VLOAD     F0,F1←A ()
VCOPY     F2,F3←F1
VCOMP     M0,M1,～M6←F0.GT.#0.0
VLOADM    F4←B () BY M0
VSUBM     F6←F2,F4 BY M1
VSTOREM   D () ←F6 BY M2
VLOADM    F5←C () BY !M3
VMULM     F7←F3,FR2 BY !M4
VADDM     F8←F5,F7 BY !M5
VSTOREM   D () ←F8 BY !M6
```

(a) マスク付きベクトル演算方式

```
MOVE      VLR←#100
VLOAD     F0←A ()
VCOMP     M0,M1,M2←F0.GT.#0.0
VGENASM   F2,F3←GR2,M0
VGENASM   F4,F5←GR2,! M1
VLOADIX   F6←A (F2)
VLOADIX   F8←B (F3)
VSUB      F10←F6,F8
VLOADIX   F7←A (F4)
VLOADIX   F9←C (F5)
VMUL      F11←F9,FR2
VADD      F12←F7,F11
VMERGEM   F13←F10,F12 BY M2
VSTORE    D () ←F13
```

(b) リストベクトル・アクセス方式

図6. プログラム1のオブジェクト・コード

するデータ依存を保証する範囲内で並列動作可能である.

4. 4 ベクトル命令実行停止

図4の例を用いて, 従来ベクトル化不可能であった「ループ外への飛び出しがあるループ」をベクトル化可能とするベクトル命令実行停止機能について説明する.

① まず, ⓐのMOVE命令でベクトル長を設定する.

② VLOAD命令ⓑおよびⓒは, ベクトルCおよびDをFIFOレジスタF0およびF1にそれぞれロードする. ⓓのVADD命令はベクトルCとDを加算する. これら命令ⓑ～ⓓは接尾に「. w」が表記されている通り, ベクトル実行停止モード下で実行される.

③ ⓔのVWHILE命令ではベクトルA (F2) に対して比較演算を行う. 比較結果が真の場合, それをすべてのデスティネーションであるマスクFIFOレジスタM0, M1, M2に出力する. 逆に, 比較結果が偽の場合, EOSをすべてのデスティネーションに出力して, 実行を完了する. これと同時に, ベクトル実行停止モード下で実行中であった命令ⓑ～ⓓは, その実行を停止させられる.

④ マスク付きベクトル演算およびロード／ストア命令ⓕ～ⓗは, それぞれのソース・マスクFIFOレジスタに従ってマスク付き実行を行う. 命令の完了は, 所定のベクトル長に達したか, あるいは, ソースマスクFIFOレジスタからEOSを取り出したときである.

5 各種ベクトル化方式に関する『順風』の性能評価

マスク付きベクトル演算方式, ベクトル分配／併合方式およ

びリストベクトル・アクセス方式に関する『順風』の性能をシミュレーションにより評価する.

5. 1 準備

(1) シミュレータ

2. 3節で述べた諸元に基づいて『順風』の動作を機能レベルでシミュレーションするソフトウェア・シミュレータ[4]を用いる. 入力は, 『順風』の機械命令（スカラおよびベクトル命令）である.

(2) ベンチマーク・プログラム

IF文の制御下にある命令数の相違による性能の変化を見るため, 図2 (a), 図5 (a), (b) の3つのベンチマーク・プログラムを用いる.

プログラム1 (図2 (a) 参照) に対して, ベクトル分配／併合方式, マスク付きベクトル演算方式, リストベクトル・アクセス方式に従ってそれぞれコンパイルした結果を図2 (b), 図6 (a), (b) に示す.

プログラム2 (図5 (a) 参照) に対するコンパイル結果は, プログラム1のオブジェクト・コードに1個のベクトル乗算命令（VMUL命令またはVMULM命令）を追加したものとなる.

プログラム3 (図5 (b) 参照) に対して, ベクトル分配／併合方式, マスク付きベクトル演算方式, リストベクトル・アクセス方式に従ってそれぞれコンパイルした結果を図7 (a), (b), (c) に示す.

(3) ベクトル命令 − 実パイプライン対応関係

各ベクトル命令を実行する実パイプラインについては, 以下のように仮定する.

- 実加減算パイプライン：ベクトル加減算命令, VCOMP命令.
- 実乗除算パイプライン：ベクトル乗除算命令, VCOPY命令, VGENASM命令, VMERGEM命令, VSWITCHM命令.
- 実ロード／ストア・パイプライン：ベクトル・ロード／ストア命令. ただし, 実ロード／ストア・パイプラインは2本存在するが, 各命令は終始同一の実ロード／ストア・パイプラインにおいて実行される.

(4) 計測値

IF文の真率を0～100%の間で変化させて, プログラムの実行に要したサイクル数（実行時間）を計測する.

5. 2 予備評価

3. 4節で, 各種ベクトル化方式における所要処理量を与えた（表1参照）. これに従って, 各々のベンチマーク・プログラムの実行に必要なベクトル要素処理量の総数を求めた. これを表2に示す. nはベクトル長を表わす. しかし, この値はすべての処理が逐次的に行われる場合に相当しており, 複数のパイプラインを備える『順風』の実行時間を直接予測するものではない. このことは, ベクトル命令とそれを処理するパイプラインとの対応関係が性能に影響を与えることを意味する.

そこで, 『順風』におけるベクトル命令 − 実パイプライン対応関係に従って, 各実パイプラインにおける延べのベクトル要素処理量を求めた. その結果を表3に示す. 表3の※印をつけた項が, 各プログラムにおける処理量の最大値, つまり, 実行時間の支配項となる. 項中のnは定数（n＝100）であるので, 各支配項は真率fを変数とする方程式となる. この式の計算値を図8に示す.

5. 3 シミュレーション結果

各ベンチマーク・プログラムに対してシミュレーションを行った結果の計測値を図8に示す.

プログラム1は, 真率に関わらず, ベクトル分配／併合方式, マスク付きベクトル演算方式, リストベクトル・アクセス方式の順に性能が良い (図8 (a) 参照). この結果は5. 2節で行った予備評価結果と一致していることが, 図よりわかる.

```
MOVE      VLR←#100
VLOAD     F0,F1←A ()
VCOMP     M0,M1,M2←F0.GT.#0.0
VSWITCHM  F2,FR1←F1 BY M0
VCLOADM   F4←B () BY M1
VMUL      F3←F2,FR2
VADD      F5←F3,F4
VSSTOREM  D ()←F9,M2
```

(a) ベクトル分配／併合方式

```
MOVE      VLR←#100
VLOAD     F0,F1←A ()
VCOMP     M0,M1,M2,M3←F0.GT.#0.0
VLOADM    F2←C () BY M0
VMULM     F3←F1,FR2 BY M1
VADDM     F4←F2,F3 BY M2
VSTOREM   D ()←F4 BY M3
```

(b) マスク付きベクトル演算方式

```
MOVE      VLR←#100
VLOAD     F0←A ()
VCOMP     M0,M1←F0.GT.#0.0
VGENASM   F2,F3←GR2,M0
VLOADIX   F4←A (F2)
VLOADIX   F5←C (F3)
VMUL      F6←F4,FR2
VADD      F7←F5,F6
VSSTOREM  D ()←F7,M1
```

(c) リストベクトル・アクセス方式

図7. プログラム3のオブジェクト・コード

表2. 延べベクトル要素処理量

プログラム	ベクトル分配／併合	マスク付きベクトル演算	リストベクトル・アクセス
1	8n−nf	7n	7n−nf
2	9n−2nf	8n	8n−2nf
3	3n+2nf	5n	2n+5nf

表3. 実パイプライン毎の延べベクトル要素処理量

プログラム	実パイプラインの種類	ベクトル分配／併合	マスク付きベクトル演算	リストベクトル・アクセス
1	加減算	2n	3n ※	2n
	乗除算	3n−nf ※	2n	4n−nf ※
	ロード／ストア	3n	5n ※	4n
2	加減算	2n	3n ※	2n
	乗除算	4n−2nf ※	3n	5n−2nf ※
	ロード／ストア	3n	5n ※	4n
3	加減算	n+nf	2n ※	n+nf
	乗除算	n+nf	n	n+nf
	ロード／ストア	3n ※	3n ※	2n+2nf ※

※：支配項

次に，プログラム1に対してベクトル乗算命令を1個追加しただけのプログラム2に対する結果は，プログラム1の結果と大きく異なる（図8（b）参照）．この結果も，5．2節で行った予備評価結果と一致している．すなわち，ベクトル分配／併合方式とリストベクトル・アクセス方式の両方式は，ベクトル乗算命令の追加により支配項が変わった．一方，マスク付きベクトル演算方式の支配項には変化がなかったからである．その結果，真率f＜0．5の範囲では，マスク付きベクトル演算方式の方がベクトル分配／併合方式より性能が良くなっている．

最後に，プログラム3の結果においては，先の2つの結果と異なり，ベクトル分配／併合方式に真率に起因する変化が見られない．これもやはり，5．2節で行った予備評価結果と一致している．

6 おわりに

以上，現在開発中であるシングルチップ・ベクトルプロセッサ・プロトタイプ『順風』における「IF文を含むDOループ」の対処法を述べた．従来から完全ベクトル化可能であったループに対しては，一般的なマスク付きベクトル演算方式，ベクトル圧縮／伸長方式およびリストベクトル・アクセス方式に加えて，“ベクトル分配／併合”方式によるベクトル化を可能としている．また，これまでは完全ベクトル化不可能であった「ループ外への飛び出しのあるループ」に対しては，“ベクトル命令実行停止”機能を導入することでベクトル化可能とした．さらに，部分ベクトル化可能なループに対しては，“ベクトルースカラ協調処理”によりベクトルユニットとスカラユニット間のオーバラップ処理を許している．

また，マスク付きベクトル演算方式，ベクトル分配／併合方式およびリストベクトル・アクセス方式に関する『順風』の性能をシミュレーションにより評価した．この結果，ベクトル命令と実パイプラインとの対応関係が性能に大きな影響を与えることが判明した．この点については，今後詳細な検討が必要である．

今後は，本稿では評価を行わなかった“ベクトルースカラ協調処理”および“ベクトル命令実行停止”機能についても，評価および検討を進めて行く予定である．

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用—，”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）．

[3] 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』－IF文を含むDOループの処理－，”情処40全大論文集，7L-4（1990年3月）．

[4] 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』の構成と性能評価，”並列処理シンポジウムJSPP'90，pp.201-208（1990年5月）．

[5] 原ほか：“『新風』プロセッサの条件分岐方式，”情処40全大論文集，7L-6（1990年3月）．

[6] 富田眞治：並列計算機構成論，第6章，pp. 141-194，昭晃堂（1980年）．

[7] Hennessy J. L. and Patterson, D. A. : Computer Architecture : A Quantitative Approach, Morgan Kaufmann Publishers, Inc., pp.350-401, 1990.

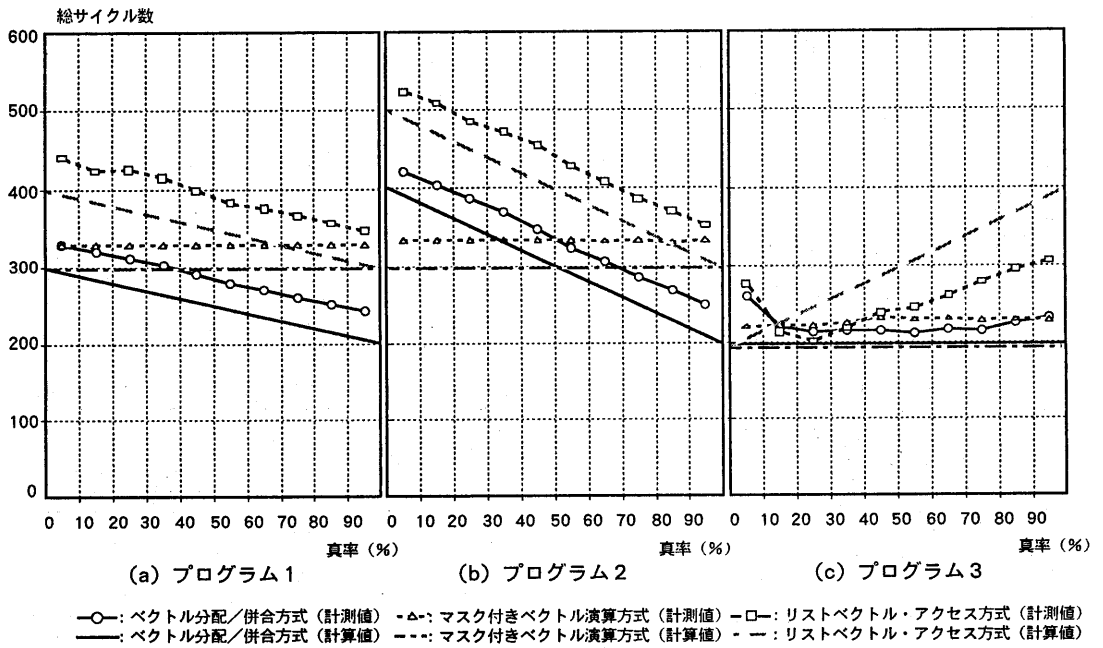

図8．評価結果