ストリーム FIFO 方式に基づくベクトルプロセッサ『順風』†

——ストリーム FIFO 方式および仮想パイプライン方式の実現法に関する評価——

弘中哲夫†† 岡崎恵三††
村上和彰†† 富田眞治††

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”を提案している．本方式は，ベクトル・レジスタとして FIFO レジスタを用い，ベクトル命令をチェイニングしてベクトル・ロード→複数のベクトル演算→ベクトル・ストアを一連の流れとして処理する点に特長がある．このとき，ベクトル演算命令を実行する演算パイプラインをできるだけ多く備える必要がある．これは，チェイニング可能なベクトル演算命令の増加は，演算命令当りのメモリ・アクセス回数の低減につながり，ひいてはベクトル演算性能の向上に寄与するからである．しかし，ハードウェア制約上，実装できる演算パイプライン数には限界がある．そこで，この演算パイプライン数に関するトレードオフ問題に対処するために，「パイプライン共有 MIMD」の概念に基づく“仮想パイプライン方式”を導入した．その結果，ベクトル命令のディスパッチ対象である“仮想パイプライン”と実在する“実パイプライン”との間の対応付けを動的に行う「実パイプライン割当てアルゴリズム」を開発する必要が生じた．本稿では，6種類のアルゴリズムを開発して，それらの性能をシミュレーションにより評価している．シミュレーション・モデルには，筆者らが開発中のストリーム FIFO 方式に基づくベクトル・プロセッサ『順風』を用いた．

1. はじめに

今日の商用スーパコンピュータでは，パイプライン処理によるベクトル演算方式を採用するのが主流となっている[10]．ベクトル演算方式は，ベクトル演算のオペランドであるベクトル・データをメモリおよびレジスタのいずれに置くかで，次の3方式に分類される．

①メモリ-メモリ演算方式：CDC Star-100, CDC Cyber 205[5] 等で代表される方式で，2つのソース・オペランドと1つのデスティネーション・オペランドいずれもメモリ上においてベクトル演算を行う．

②レジスタ-レジスタ演算方式：Cray-1[4),7)] で代表される方式で，2つのソース・オペランドと1つのデスティネーション・オペランドいずれもレジスタ(ベクトル・レジスタと呼ぶ)上においてベクトル演算を行う．ベクトル演算の前後に，ベクトル・データのレジスタへ/からのロード/ストアが必要で，“ロード/ストア・アーキテクチャ”とも呼ぶ．Cray X-MP[7)]，Cray-2[7)]，富士通 VP[11)]，日立 S[12),13)]，日電 SX[14)] 等，大部分のベクトル・プロセッサがこの方式を採用している．

③レジスタ-メモリ演算方式：上記①と②の折衷方式で，1つのソース・オペランドと1つのデスティネーション・オペランドはベクトル・レジスタ上に，残りのソース・オペランドはメモリ上においてベクトル演算を行う．IBM 3090 VF[9)] に採用されている．

メモリ-メモリ演算方式およびレジスタ-レジスタ演算方式のいずれにも，各々長所短所がある[1)]．ただ，現在の商用スーパコンピュータの動向を見る限りでは，レジスタ-レジスタ演算方式の方に完全に軍配が上がったと言えよう．しかし，レジスタ-レジスタ演算方式の最大の短所である「有限長のベクトル・レジスタに起因するオーバヘッド」に対しては，効果的な解がまだ与えられていない．

そこで我々は，レジスタ-レジスタ演算方式の長所を損なうことなく，その短所である「有限ベクトル・レジスタ長」問題を解決することを主目的として，“ストリーム FIFO 方式 (Streaming/FIFO Architecture)”と呼ぶベクトル演算方式を提案している[1)]．本方式は，上記の分類法で言うと“FIFO レジスタ-FIFO レジスタ演算方式”となる．すなわち，2つのソース・オペランドと1つのデスティネーション・オ

† A Vector Processor Based on Streaming/FIFO Architecture —Evaluation for Implementing Streaming/FIFO Architecture and Virtual Pipeline— by TETSUO HIRONAKA, KEIZOU OKAZAKI, KAZUAKI MURAKAMI and SHINJI TOMITA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科情報システム学専攻

ペランドいずれも，FIFO 動作をするベクトル・レジスタ（FIFO レジスタと呼ぶ）上においてベクトル演算を行う．ベクトル演算の前後にはレジスタ-レジスタ演算方式同様，ベクトル・データの FIFO レジスタへ/からのロード/ストアが必要である．ただし，FIFO レジスタは従来のベクトル・レジスタと異なり，FIFO 動作により「論理的に無限長のベクトル・レジスタ」として振舞うことから，長いベクトル・データに対してもストリップ・マイニング処理を施す必要がない．この点がストリーム FIFO 方式の特長であり，メモリ-メモリ演算方式と共通の長所である．

さらに，ストリーム FIFO 方式の“ストリーム”とは，文献 8) のストリーム方式に由来するものである．すなわち，スカラ命令に対してもベクトル・レジスタへのアクセスを可能とするものである．これにより，ベクトル・ユニット-スカラ・ユニット間の柔軟な並行/協調処理を目指している．

我々は現在，ストリーム FIFO 方式に基づく最初のプロトタイプである『順風』[1)~3)] を開発している．『順風』には，『パイプライン共有 MIMD』[6)] 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．すなわち，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時分割共有する．このとき，個々の命令から見た論理的なパイプラインを“仮想パイプライン”，実在するパイプラインを“実パイプライン”と呼ぶ．仮想パイプラインを導入することにより，同時にチェイニング可能な命令数の増加を図っている．

本稿ではまず，2章および3章でストリーム FIFO 方式および仮想パイプライン方式についてそれぞれ述べる．4章で，『順風』の構成について述べる．5章で仮想パイプラインへの実パイプライン割当てアルゴリズムについて述べた後，6章でシミュレーションによる評価結果を述べる．

2. ストリーム FIFO 方式

2.1 概　　要

まず，従来のベクトル演算方式について，ベクトル長，パイプライン・チェイニング，メモリ・アクセス，スカラ・プロセッサとの並行/協調処理の4つの課題に対する最も適した方式を挙げると次のようになる．

①ベクトル長：メモリ上にベクトル演算のオペランドを置くことで1命令で処理できるベクトル長に制限を与えないメモリ-メモリ演算方式．

②パイプライン・チェイニング[4)]：チェイニングのためのデータ依存関係解析が容易なレジスタ-レジスタ演算方式．

③メモリ・アクセス：ベクトル演算の中間結果をベクトル・レジスタ上に保持でき，かつ，スタートアップ時間の短いレジスタ-レジスタ演算方式．

④スカラ・プロセッサとの並行/協調処理：ベクトル・レジスタがスカラ命令から直接アクセス可能なストリーム方式[8)]．

以上の点に鑑み，まず①②③については，メモリ-メモリ演算およびレジスタ-レジスタ演算の両方式の長所を活かすために，従来のベクトル・レジスタに FIFO 動作をさせる“FIFO レジスタ-FIFO レジスタ演算方式”を導入する．さらに④については，FIFO レジスタにスカラ命令がアクセスすることを許して，“ストリーム方式”を融合する．この結果得られたのが，“ストリーム FIFO 方式”である．

ストリーム FIFO 方式は，以下の特長を持つ．

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 動 作 説 明

図1の例を用いて，ストリーム FIFO 方式に基づくベクトル演算動作を説明する．図1(a)は二次回帰演算を行うソース・コードであり，図1(b)がストリーム FIFO 方式のためのオブジェクト・コードである．なお，F0～F5 は FIFO レジスタを示す．

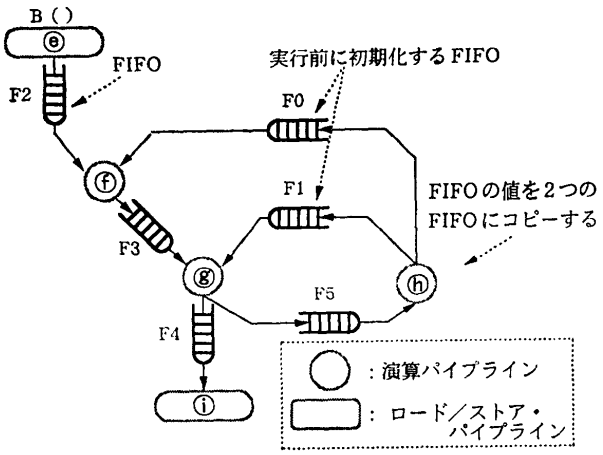
①まず，F0と F1 の初期化を行う．MOVE 命令ⓐでベクトル長レジスタ（VLR）を2に設定し，VLOAD 命令ⓑで F1 に A(0) と A(1) をベクトルロードする．次に LOAD 命令ⓒで F0 に

```
DO 10 I=2, 100
   A (I) =B (I) *A (I-1) +A (I-2)
10 CONTINUE
```

(a) ソースコード
(a) Source code.

```
MOVE    VLR ← #2          ⓐ
VLOAD   F1  ← A (0)       ⓑ
LOAD    F0  ← A (1)       ⓒ
MOVE    VLR ← #98         ⓓ
VLOAD   F2  ← B (2)       ⓔ
VMUL    F3  ← F2,F0       ⓕ
VADD    F4,F5 ← F3,F1     ⓖ
VCOPY   F0,F1 ← F5        ⓗ
VSTORE  A (2) ← F4        ⓘ
```

(b) オブジェクトコード
(b) Object code.



(c) データフロー・グラフ
(c) Dataflow graph.

図1 ストリーム FIFO 方式を用いたベクトル処理
Fig. 1 Example of vector processing with Streaming/FIFO Architecture.

A (1) をロードする.

②次にループの本体を実行する. ⓓの MOVE 命令によりベクトル長レジスタを 98 に設定し, ベクトル命令ⓔⓕⓖⓗⓘを実行する. これにより, パイプライン (ベクトル命令) をノード, FIFO レジスタを枝とするデータフロー・グラフ (図1(c)参照) が生成される.

③このグラフに基づき, ベクトル処理は次のように進む. VLOAD 命令ⓔで, 配列Bを第2要素から F2 にベクトルロードする. VMUL 命令ⓕは, F2 のデータと F0 のデータとを乗算し, 結果を F3 に格納する. VADD 命令ⓖは, F3 のデータと F1 のデータとを加算し, 結果を F4 と F5 の双方に格納する. F5 に格納されたデータは, VCOPY 命令ⓗによりそのまま F0 と F1 にコピーされ, 引続き命令ⓕⓖに用いられる. 一方, F4 に格納されたデータは, VSTORE 命令ⓘによりメモリにベクトル・ストアされる.

3. 仮想パイプライン方式

3.1 要　　件

ストリーム FIFO 方式では複数の資源 (演算パイプライン, ロード/ストア・パイプライン, FIFO レジスタ) の資源はすべて結合網で結ばれ, 演算パイプラインおよびロード/ストア・パイプラインをノードとし, FIFO レジスタを枝とした任意のデータフロー・グラフを構成できるようになっている. ベクトル演算はこのデータフロー・グラフに従って行われる.

このとき, パイプライン数に関するトレードオフが課題になる. すなわち, ストリーム FIFO 方式でベクトル演算を行う場合, 必ずベクトル・ロード→ベクトル演算 (複数) →ベクトル・ストアとパイプラインをチェイニングしてベクトル演算を行う. よって, 一時にチェイニング可能な演算パイプライン数が少ないと, 演算命令当りのロード/ストア回数が増しメモリ競合の増加を招く. したがって, 一時になるべく多くのベクトル命令をチェイニングするよう, できるだけ多くの演算パイプラインを設けた方がよい. しかし, パイプライン数をいたずらに増やすと, ハードウェアコストの増大を招くことになる.

3.2 パイプライン共有 MIMD

上記の課題に対処するため, 『順風』では「パイプライン共有 MIMD」[6] の概念を取り入れた. すなわち, 1個の命令が独占的に1つ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. このとき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.

仮想パイプラインの導入により, 以下の効果が得られる.

①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, (回帰型演算およびスカラ・プロセッサとの並列/協調処理における) 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が可能となる．

本方式採用により実パイプラインと仮想パイプライン間の割当てを如何に行うかという課題が生じる．この課題については5章で述べる．

4.『順　風』

『順風』はスカラ命令を処理するスカラ・ユニットとベクトル命令を処理するベクトル・ユニットからなる．本稿では『順風』のベクトル・ユニットのみについて述べる．**図2**にベクトル・ユニットの構成を示す．『順風』のベクトル・ユニットは次の6つから構成される．

（1）ストリーム制御ステーション・ユニット

図3に示すストリーム制御ステーション（SS：Streaming Station）を仮想パイプライン数だけ備える．1個のSSは，1本の仮想ロード/ストア・パイプラインないし仮想演算パイプラインに対応する．つまり，SSは仮想パイプラインの実体であり，ベクトル命令のディスパッチ対象である．SSは，ディスパッチされた命令の実行環境を保持すると同時に命令の実行制御を行う．

命令の実行制御を司るステートマシンを**図4**に示す．本ステートマシンは，オペランドFIFOレジスタおよびベクトル長カウンタの状態に従って，次のようにステート遷移を行う．まず，FIFO待ちステートにおいてオペランドFIFOレジスタの状態をチェックし，命令実行可能であれば実パイプライン割当てユニット（後述）に対して実パイプライン獲得要求を出して実パイプライン待ちステートへと遷移する．実パイプラインが割り当てられると，実行ステートに遷移して命令実行を行う．実行ステート中も

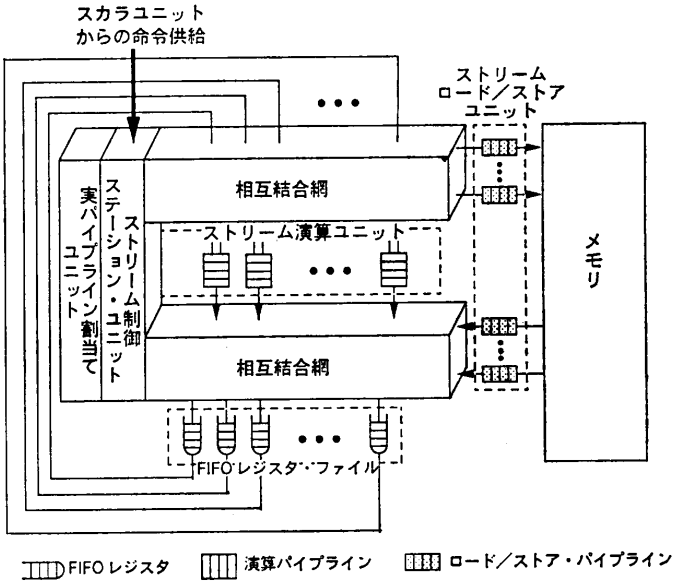
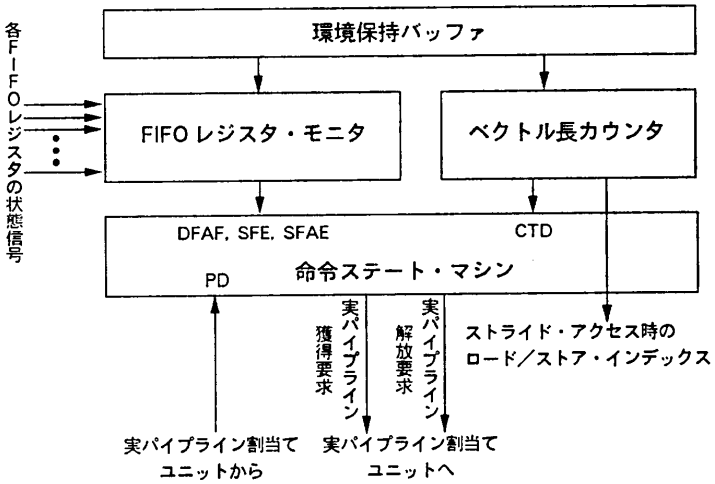


図2『順風』ベクトル・ユニットの構成
Fig. 2 Vector unit of【dʒúmpu:】.



CTD ： CompleTeD
命令が指定された回数実行された時「真」
DFAF： Destination Fifo Almost Full
これ以上の命令実行はFIFOレジスタのオーバーフローを招く時「真」
SFE ： Source Fifo Empty
データがFIFOレジスタ内に存在しない時「真」
SFAE： Source Fifo Almost Empty
命令実行を行うとFIFOレジスタが空になる状態の時「真」
PD ： Pipline Dispatched
パイプラインが割り当てられた時「真」
（※）FIFOレジスタとはデータFIFOレジスタおよびマスクFIFOレジスタを含めたものである

図3 ストリーム制御ステーションの構成
Fig. 3 Streaming Station.

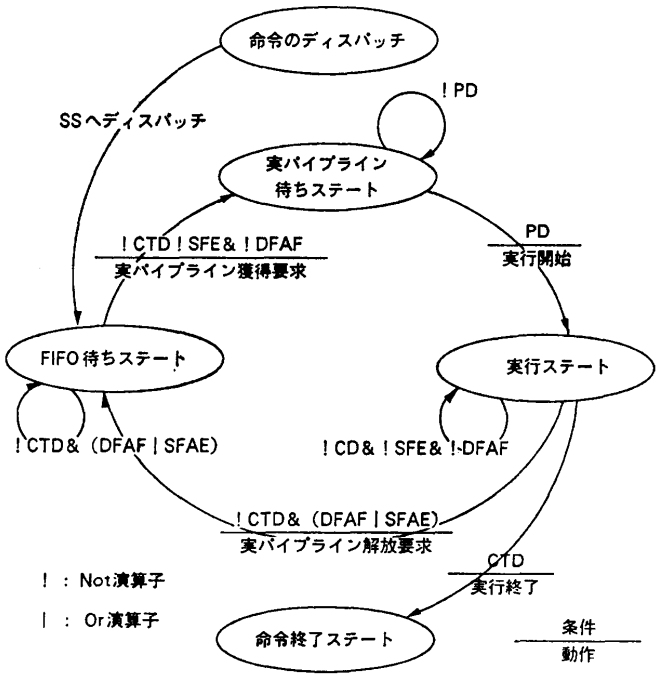


図 4 命令ステート・マシン
Fig. 4 Instruction state machine.

オペランド FIFO レジスタの状態をチェックしており，実行継続不可能になると実パイプライン解放要求を出し再び FIFO 待ちステートに遷移する．ベクトル長カウンタで指定されたベクトル長を越えたら，命令終了ステートに遷移して命令実行を完了する．

（2） 実パイプライン割当てユニット

SS から出された実パイプライン獲得/解放要求に応えて，実パイプラインの割当て/解放を行う．同時に複数の獲得要求がある場合は，これらの間の調停を行う．ここで行う実パイプライン割当てのアルゴリズムに関しては，5 章および 6.3 節を参照されたい．

（3） ストリーム・ロード/ストア・ユニット

1 本以上の実ロード/ストア・パイプラインを備える．

（4） ストリーム演算ユニット

1 本以上の実演算パイプラインを備える．

（5） FIFO レジスタ・ファイル

16 本のデータ FIFO レジスタおよび 16 本のマスク FIFO レジスタを備える．各 FIFO レジスタは，相互結合網により実演算パイプラインおよび実ロード/ストア・パイプラインに接続される．

（6） 相互結合網

ストリーム・ロード/ストア・ユニットおよびストリーム演算ユニットを FIFO レジスタ・ファイルに結合する．

5. 実パイプライン割当てアルゴリズム

5.1 要　　件

仮想パイプラインに実パイプラインを割り当てる際の要件として，次の 3 つが存在する．

（1） 実パイプラインの均等割当て

チェイニングされた仮想パイプラインは，FIFO レジスタを枝，仮想パイプラインをノードとするデータフロー・グラフを形成する．グラフ内のすべての仮想パイプラインが見かけ上並列にベクトル命令を実行しているとき，スループットは最大になる．これには，特定の仮想パイプラインに集中することなく均等に実パイプラインを割り当てる必要がある．

（2） 実パイプライン割当て時のオーバヘッド抑止

実パイプラインと仮想パイプラインとの対応を切り換える時，以下のオーバヘッドが発生する．サイクル・タイムの制約が厳しいベクトル・プロセッサでは，これらのオーバヘッドを極力抑える必要がある．

①相互接続網上でのリンクの設定/解放オーバヘッド

②割当てアルゴリズム実行に伴うオーバヘッド (6.3 節で詳述)．

（3） ハードウェア化

当然ながら，ハードウェア化に際しては，以下の用件を満足しなくてはならない．

①簡単な構成

②高速な実現が可能

これらの要件に鑑み，次の 3 種類の割当てアルゴリズムを検討する．

①固定優先順位方式 (5.2 節)

②ラウンドロビン方式 (5.3 節)

③トークン・リング方式 (5.4 節)

5.2 固定優先順位方式

命令が SS にディスパッチされる時に固定的な優先順位を仮想パイプラインにつけ，実パイプラインを仮想パイプラインに割り当てる際の優先順位とする．この優先順位は命令終了まで変化しない．図 5 に示すよ

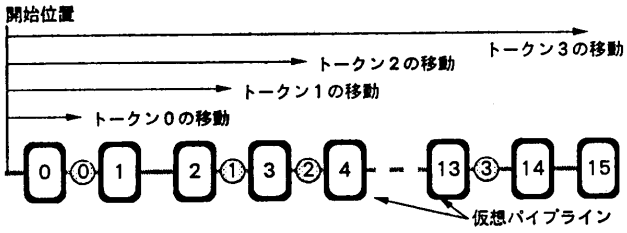


図 5 固定優先順位方式
Fig. 5 Fixed priority approach.

うに割当ては常に優先順位の高い仮想パイプラインから行われる．ここで実パイプラインの使用権をトークンと呼ぶ．優先順位の低い仮想パイプラインに対する実パイプラインの割当ては，優先順位の高い仮想パイプラインのFIFOレジスタにデータが存在しない場合あるいは，仮想パイプラインの生成する結果を FIFO レジスタに格納できない場合のみ行われる．

固定優先順位方式は，割り当てられた実パイプライン解放の方法により，さらに次の2つに分類できる．

①量子時間消費による解放：規定のサイクル数だけ実パイプラインを使用した後，実パイプラインの解放を行う．

②FIFO レジスタの状態による解放：FIFO レジスタがアンダフロー/オーバフローしそうになったら，トークン（実パイプライン）を解放する．

固定優先順位方式の問題点は，命令間の優先順位が命令を SS にディスパッチした時に決定され，命令終了時まで変更されない．よって，不均等な割当てが行われる可能性がある．しかし，仮想パイプラインの優先順位を変更する機構を必要としないといった利点を有する．

5.3 ラウンドロビン方式

仮想パイプラインに対する均等な実パイプラインの割当てを満足する割当てアルゴリズムの1つとして，ラウンドロビン方式がある．ラウンドロビン方式では，循環待ち行列につながれた実行待ち仮想パイプラインに対して，次のように実パイプラインを割り当てる．

①実行可能となった仮想パイプラインを待ち行列の最後尾につなぐ．

②実パイプライン割当ての順番が来た実行待ち仮想パイプラインに対して，一時には1量子時間だけ割当て時間を与える．

③実パイプラインを割り当てられた仮想パイプラインは，ソース FIFO レジスタがアンダフロー，または，デスティネーション FIFO レジスタがオーバフローしそうになると実パイプラインを解放する．

④あるいは，1量子時間を消費した仮想パイプラインは，実パイプラインを解放し，再び待ち行列の最後尾につながれる．

ラウンドロビン方式の待ち行列は循環待ち行列となっており，公平な実パイプラインの割当てが実現される．しかし，本方式を実現する上での問題として，循環待ち行列を高速なハードウェアで実現するのが困難である点が挙げられる．すなわち，複数の実パイプラインにおいて異なるタイミングで切換えが発生するため，待ち行列における割当て順番の入れ替わりが発生する．この問題を回避し，かつ，ハードウェアによる実現を容易にするため，次に述べるトークン・リング方式を検討する．

5.4 トークン・リング方式

図6に示すように，仮想パイプラインは概念上1本のリングに連結されており，そのリング上を実パイプラインに対応する複数個のトークンが流れているものとする．トークン・リング方式では次のように実パイプラインを割り当てる．

①実行可能となった仮想パイプラインはリング上のいずれかのトークンを獲得しようとする．

②トークンを獲得したら，対応する実パイプラインが割り当てられる．

トークン・リング方式はトークン解放の方法によりさらに次の2つに分類できる．

①量子時間消費による解放：ラウンドロビン方式と

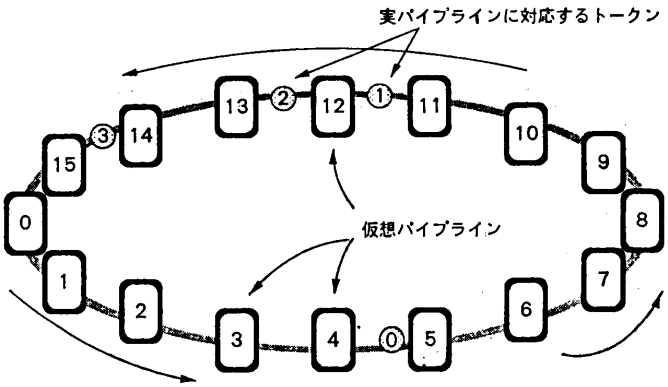


図 6 トークン・リング方式
Fig. 6 Token ring approach.

同様に量子時間の消費で実パイプラインの解放を行う．

②FIFO レジスタの状態による解放：FIFO レジスタがアンダフロー/オーバフローしそうになったら，トークン（実パイプライン）を解放する．

6. シミュレーションによる評価[2)]

6.1 シミュレータ

5章で述べた実パイプライン割当てアルゴリズム，および，FIFO レジスタ長が性能に与える影響をシミュレーションにより評価した．シミュレーション・モデルとして，以下のハードウェア構成を採る『順風』を想定した．

- 実演算パイプライン数：4本
- 実ロード/ストア・パイプライン数：6本
- ストリーム・ステーション（SS）数：
  - —仮想演算パイプライン用：16 個
  - —仮想ロード/ストア・パイプライン用：16 個
- FIFO レジスタ数：16 本
- メモリ・バンク数：128 バンク

評価基準としては実演算パイプライン，および，実ロード/ストア・パイプラインの使用率を用いた．使用率は次式により計算した．

$$\text{使用率} = \frac{\text{全オペレーション数}}{(\text{実行所要サイクル数} \times \text{実パイプライン数})}.$$

6.2 ベンチマーク・プログラム

ベンチマーク・プログラムとして，Laurance Livermore Kernel の LLK 1, LLK 2, LLK 3, LLK 4, LLK 5, LLK 7, および，深さ3の完全2分木型のデータフロー・グラフを持つプログラム PYRAMID を使用した．プログラムはハンド・コンパイルにより『順風』アセンブラ・コードに変換したものを使用した．

6.3 評価した実パイプライン割当てアルゴリズム

5章で述べた各方式の長所短所に鑑みて，固定優先順位方式およびトークン・リング方式に基づく実パイプライン割当てアルゴリズムを評価の対象とする．具体的には，実現の際のハードウェア構成を考慮に入れて，次の2つの異なる割当て方針を前提とした計6種類のアルゴリズムを評価する．

①完全多対多割当て：割当て要求を出している仮想パイプラインおよび割当て可能な実パイプラインがそれぞれ複数本存在する場合，両者間の多対多割当てを行う必要がある．このとき，一時に完全な多対多割当てを行うことを保証する．すなわち，2本以上の実パイプラインが1本の仮想パイプラインに同時に割り当てられるといった無駄な割当てを防ぐ．実パイプライン割当ての効率は高くなるが，本割当て方針は基本的に逐次性が強いので，高速な実現のためには莫大なハードウェア量を必要とする．

②不完全多対多割当て：上記とは異なり，2本以上の実パイプラインが1本の仮想パイプラインに同時に割り当てられるといった無駄な割当てが起こり得る．この場合，いずれかの実パイプライン1本のみが割り当てられ，他の実パイプラインの割当ては次サイクルに延期される．これを以後，“割当ての衝突”と呼ぶ．このように，実パイプライン割当ての効率は上記方針に比べると低くなる可能性があるが，比較的少量のハードウェアで高速な実現が可能である．これは，実パイプラインごとに独立かつ並列に仮想パイプラインの選択が行えるからである．

（1） 割当て方針：完全多対多割当て

①FP/TS (Fixed Priority/Time Slicing)：固定優先度順位方式に基づく割当てアルゴリズムである．1回の割当てにつき1量子時間（1マシンサイクル）だけ実パイプラインを割り当てる．

②TR/TS (Token Ring/Time Slicing)：トークン・リング方式に基づく割当てアルゴリズムであり，1回の割当てにつき1量子時間（1マシンサイクル）だけ実パイプラインを割り当てる．

③TR/DD (Token Ring/Data Driven)：トークン・リング方式に基づく割当てアルゴリズムであり，FIFO レジスタの状態により実パイプラインの解放を行う．したがって，割り当てられた実パイプラインに対するデータの供給および格納が滞らなければ何サイクルでも実パイプラインを獲得し続ける．

（2） 割当て方針：不完全多対多割当て

④FP/DD (Fixed Priority/Data Driven)：固定優先度順位方式に基づく割当てアルゴリズムである．FIFO レジスタの状態により実パイプラインの解放を行う．割当ての衝突が生じた場合，調停により優先順位の高い実パイプラインを割り当てる．割当てから外れた実パイプラインは，当該サイクルでの割当てを締める．

⑤TR/DD/N (TR/DD/Next)：TR/DD アルゴリズムに基づく．割当ての衝突が生じた場合，調停により優先順位の高い実パイプラインを割り当てる．割

当てから外れた実パイプラインの次サイクルにおけるトークン・リング上の位置は，衝突を起こした仮想パイプラインの次の仮想パイプラインとする．

⑥ TR/DD/F (TR/DD/First)：TR/DD アルゴリズムに基づく．割当ての衝突が生じた場合，調停により優先順位の高い実パイプラインを割り当てる．割当てから外れた実パイプラインの次サイクルにおけるトークン・リング上の位置は，あらかじめ定められた初期位置とする．この初期位置はトークン・リングを実パイプライン数で等分割した位置に相当し，実パイプラインごとに異なる．

6.4 実パイプライン割当てアルゴリズムに関する評価

実演算パイプラインおよび実ロード/ストア・パイプラインそれぞれに対して，6種類の実パイプライン割当てアルゴリズムを適用し，計 36 通りの組合せについてシミュレーションを行った．

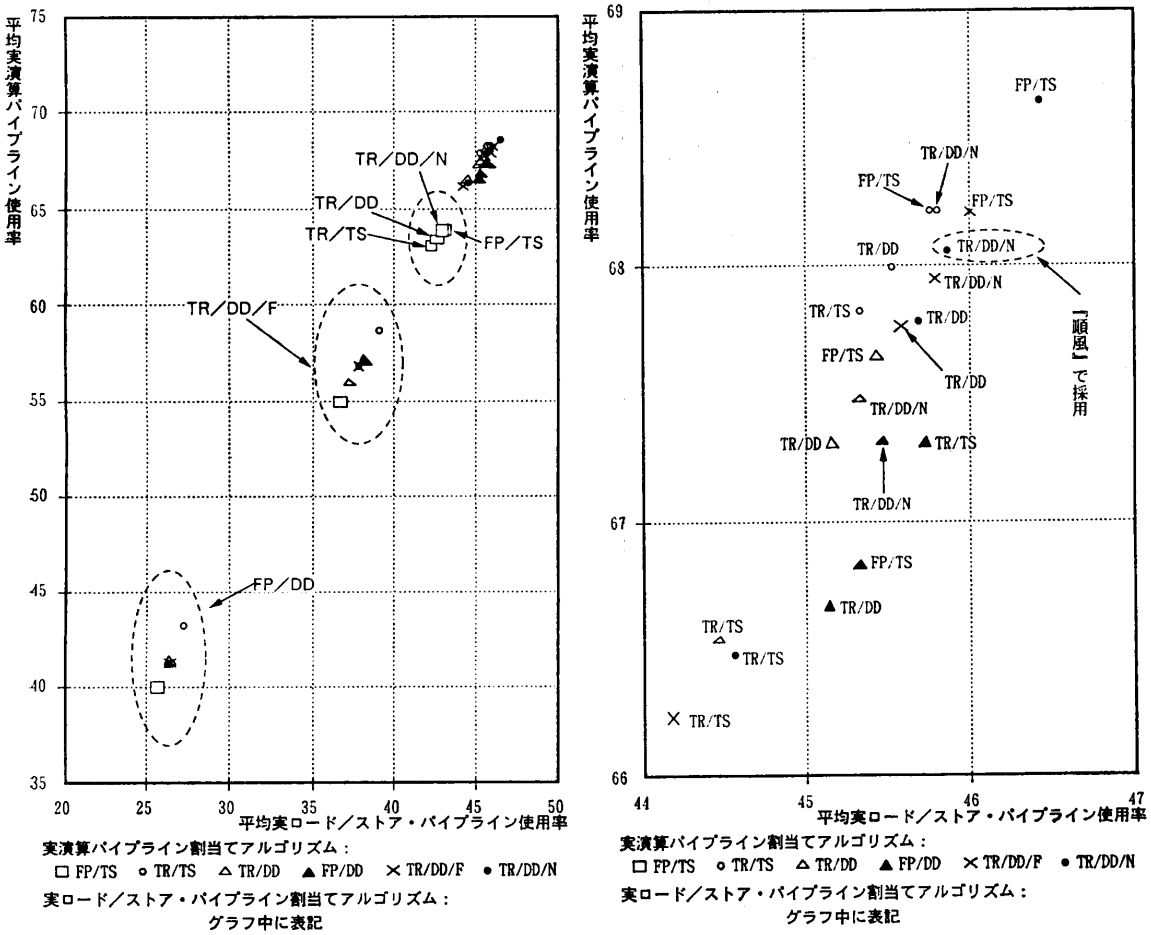
なお，ここでの評価に当たっては，FIFO レジスタ長を 32 としてシミュレーションを行った．

図7にシミュレーション結果を示す．図中の値は，すべてのベンチマーク・プログラムに対するパイプライン使用率の算術平均値を表す．X 軸は実ロード/ストア・パイプライン使用率を，Y 軸は実演算パイプライン使用率をそれぞれ示す．

図7(b)より，最も効率の良い実パイプライン割当てアルゴリズムは，

- 演算パイプライン：TR/DD/N
- ロード/ストア・パイプライン：FP/TS

の組合せであることがわかる．しかし，FP/TS の割当て方針は“完全多対多割当て”であり，その高速な



(a) 全体図
(a) Full scale.

(b) トップグループの拡大図
(b) Magnified top group.

図7 実パイプライン割当てアルゴリズムの評価
Fig. 7 Evaluation for real-pipeline dispatch algorithms.

実現は多大なハードウェアを要する．そこで，割当て方針として“不完全多対多割当て”を採るアルゴリズムにのみ注目すると，

- 演算パイプライン：TR/DD/N
- ロード/ストア・パイプライン：TR/DD/N

の組合せが最も効率良い．しかも，これら両組合せ間の実演算パイプライン使用率の差は 0.588% 程度と，ほとんど無視できる．

以上の結果から，少量のハードウェアで高速な実現が可能な実パイプライン割当てアルゴリズムとして，

- 演算パイプライン：TR/DD/N
- ロード/ストア・パイプライン：TR/DD/N

の組合せを『順風』では採用することにする．以後の評価では，この組合せを前提とする．

6.5 FIFO レジスタ長に関する評価

6.4 節で決定した実パイプライン割当てアルゴリズムに対する最適な FIFO レジスタ長を決定するため，FIFO レジスタ長と実演算パイプライン使用率との関係をシミュレーションにより調べた．この結果を図8に示す．図8より，最適な FIFO レジスタ長が 32 であることがわかる．

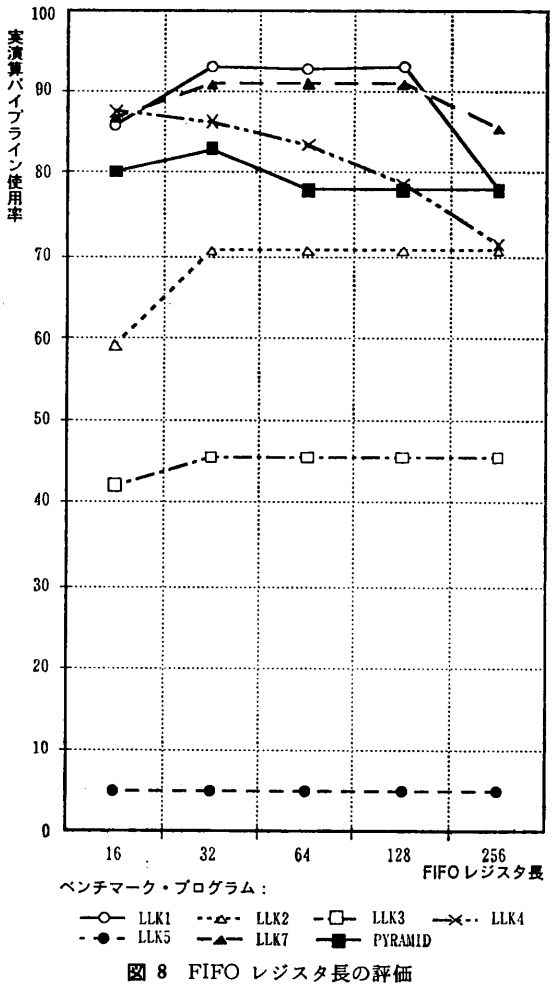


図 8 FIFO レジスタ長の評価
Fig. 8 Evaluation for FIFO-register sizes.

さらに，図8から，FIFO レジスタ長は短すぎても長すぎても実演算パイプライン使用率を低下させることがわかる．これは，FIFO レジスタ長が短い場合，FIFO レジスタのオーバフローが多発することによる．すなわち，仮想パイプラインの切換えが多発し，実パイプライン割当てのオーバヘッドが露見するためである．一方，FIFO レジスタ長が長い場合は，仮想パイプラインの切換えがなかなか起きないため，実パイプラインの割当てが不均等になるからである．

7. おわりに

以上，我々が提案しているストリーム FIFO 方式，本方式に基づく最初のプロトタイプ・プロセッサ『順風』，および，『順風』に採用した仮想パイプライン方式について述べた．ストリーム FIFO 方式は従来のベクトル演算方式に比べ，一時に処理可能なベクトル長に制限がなく，かつ，柔軟なベクトル-スカラ協調処理が可能であるという特長を有する．さらに，本方式が他のベクトル演算方式に比べて有効であるためには，パイプライン・チェイニングにより同時に実行可能な命令を十分に多くする必要がある．このために，「パイプライン共有 MIMD」の概念に基づく仮想パイプライン方式を導入した．

この仮想パイプラインと実パイプラインとの対応付けを動的に行うために，6種類の実パイプライン割当てアルゴリズムを開発し，これらをシミュレーションにより評価した．その結果，適切なアルゴリズムを選択することにより，少量のハードウェアで高い演算パイプライン使用率を達成できることが判明した．

なお，今日のスーパコンピュータに用いられているような非常に高速なクロック周波数でアルゴリズム TR/DD/N を実現する場合，アルゴリズムのパイプライン化を図る必要が生じるであろう．例えば，命令の実行可能性判定，実パイプライン割当て，および，競合の判定に，それぞれ1ステージずつ割り当てるような分割方法が考えられる．ステージ分割の最適化および性能評価は今後の研究課題である．

謝辞 現在筆者らとともに設計・開発を行っている橋本隆氏，および，日頃御討論いただく富田研究室の皆様に感謝します．

参考文献

1) 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』，信学技報，CPSY-89-39 (1989).
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』の構成と性能評価，情報処理学会並列処理シンポジウム JSPP '90 論文集，pp. 201-208 (1990).
3) 岡崎ほか：『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—，情報処理学会研究会報告，ARC-85-3 (1990).
4) Russell, R.M.: The CRAY-1 Computer System, *Comm. ACM*, Vol. 21, No. 1, pp. 63-72 (1978).
5) Lincoln, N.R.: Technology and Design Tradeoffs in the Creation of a Modern Supercomputer, *IEEE Trans. Comput.*, Vol. C-31, No. 5, pp. 349-362 (1982).
6) Jordan, H.F.: Performance Measurements on HEP-A Pipelined MIMD Computer, *Proc. 10th Int'l Symp. Computer Architecture*, pp. 207-212 (1983).
7) Thompson, J.R.(編)：CRAY-1, CRAY X-MP, CRAY-2 とその将来：Cray Research のスーパーコンピュータ，スーパーコンピュータ (Fernbach, S.(編)，長島(訳))，pp. 89-116，パーソナルメディア (1988).
8) Wulf, Wm. A.: The WM Computer Architecture, *ACM SIGARCH Computer Architecture News*, Vol. 16, No. 1, pp. 70-84 (1988).
9) Padegs, A. et al.: The IBM System/370 Vector Architecture: Design Considerations, *IEEE Trans. Comput.*, Vol. 37, No. 5, pp. 509-520 (1988).
10) 田中：500 MFLOPS 級の商用スーパコンピュータが稼働へ，日経エレクトロニクス，No. 314, pp. 108-126 (1983).
11) 平栗ほか：マシン・サイクル 7.5 ns を達成した並列パイプライン処理方式のスーパコンピュータ FACOM VP，日経エレクトロニクス，No. 314, pp. 131-155 (1983).
12) 小高ほか：最大性能が 630 MFLOPS で 1G バイトの半導体拡張記憶が付くスーパコンピュータ HITAC S-810，日経エレクトロニクス，No. 314, pp. 159-184 (1983).
13) 河辺ほか：シングル・プロセッサで最大性能 2 GFLOPS の S-820，日経エレクトロニクス，No. 437, pp. 111-125 (1987).
14) 古勝ほか：最大性能 1.3 GFLOPS，マシン・サイクル 6 ns のスーパコンピュータ SX システム，日経エレクトロニクス，No. 356, pp. 237-272 (1984).

(平成 2 年 8 月 20 日受付)
(平成 3 年 2 月 12 日採録)

弘中 哲夫（正会員）

1965 年生．1988 年山口大学工学部電気工学科卒業．1990 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．現在同大学院博士課程に在学中．計算機アーキテクチャ，並列処理システムの研究に従事．

岡崎 恵三

1967 年生．1990 年九州大学工学部情報工学科卒業．現在，同大学大学院総合理工学研究科情報システム学専攻修士課程に在学中．並列処理，ベクトル処理，計算機アーキテクチャの研究に従事．

村上 和彰（正会員）

1960 年生．1982 年京都大学工学部情報工学科卒業．1984 年同大学院工学研究科情報工学専攻修士課程修了．同年富士通(株)本体事業部に入社．主として汎用計算機M シリーズのアーキテクチャ開発に従事．1987 年九州大学工学部助手，1983 年同大学院総合理工学研究科情報システム学専攻助手，現在に至る．計算機アーキテクチャ，コンパイラ，並列処理等の研究に従事．著書「計算機システム工学（共著，昭晃堂）」．電子情報通信学会，日本応用数理学会，ACM，IEEE，IEEE-CS 各会員．

富田 眞治（正会員）

1945 年生．1968 年京都大学工学部電子工学科卒業．1973 年同大学院博士課程修了．工学博士．同年京都大学工学部情報工学教室助手．1978 年同助教授．1986 年九州大学大学院総合理工学研究科教授，1991 年京都大学工学部情報工学科教授，現在に至る．計算機アーキテクチャ，並列処理システムなどに興味を持つ．著書「並列計算機構成論」「計算機システム工学」「並列処理マシン」など．電子情報通信学会，IEEE，ACM 各会員．