Dual-Flow:
制御駆動とデータ駆動を融合したプロセッサ・アーキテクチャ

五 島 正 裕[†]　グェン ハイハー[††]
森 眞 一 郎[†]　富 田 眞 治[†]

スーパースカラの動的命令スケジューリングにはさまざまな利点があるが，VLIW にくらべてサイクル・タイムを短縮しにくいという欠点がある．本稿では，制御駆動とデータ駆動を融合した新しいアーキテクチャdual-flow を提案する．Dual-flow は，制御駆動型と同様に制御の流れを持つが，データ駆動型と同様にレジスタを持たず，命令間のデータの受け渡しを陽に指定する．この結果，スーパースカラ的な動的命令スケジューリングを VLIW 並のサイクル・タイムで実現することができる．

Dual-Flow:
A hybrid processor architecture between control- and data-driven

Masahiro Goshima,[†] Nguyen Haiha,[††] Shin-ichiro Mori[†]
and Shinji Tomita[†]

Dynamic instruction scheduling of superscalar has various merits, but it is also disadvantage respect of cycle time compared with VLIW. In this paper, Dual-Flow, a hybrid processor architecture between control- and data-driven is proposed. Dual-Flow has a flow of control like control-driven, while it doen not have registers and transfers data between instructions directly like data-driven. As a result, it realizes dynamic instruction scheduling of superscalar at a cycle time of VLIW.

1. はじめに

スーパースカラ（以下 SS と略）は，商用プロセッサとして大きな成功を収めた．その最も大きな理由の 1 つには，命令発行多重度を増やしていった場合にもバイナリ互換性を保つことができるということがある．

SS がバイナリ互換性を保てるのは，動的に命令をスケジューリングするためである．しかし，命令スケジューリングを行うハードウェアの複雑さがサイクル・タイムを短縮する上での足枷となる．

それに対して，コンパイラが静的に命令スケジューリングを済ませる VLIW プロセッサは，サイクル・タイムの点で SS より有利である．命令発行多重度が増加するにしたがってその傾向は顕著になるため，近い将来 VLIW が SS を淘汰すると見る向きもある．

しかしすべてを静的に行う VLIW には，バイナリ互換性以外にも，真に動的な擾乱に対して敏感であるという弱点がある．例えばロード命令がキャッシュ・ミスを起こしたとき，SS はその命令に依存しない命令を発見し実行を続けることができるのに対して，VLIW は依存する命令以降を一切実行しない．また，ミス時のペナルティが大きくなる傾向が強いため，積極的な投機を行いにくい．

我々は，SS の動的命令スケジューリングと VLIW のサイクル・タイムを両立する手法を模索してきた．その 1 つの解として，本稿では **Dual-Flow** と呼ぶ新しいプロセッサ・アーキテクチャを提案する．

以下 2章でまず，動的命令スケジューリングの複雑さについて考察した後，3章で dual-flow の実行モデルと実装例について説明する．4章では，予備的な性能予測を行う．

2. 動的命令スケジューリング

本章では，MIPS R10000[1)]を例に，動的命令スケジューリングの処理の複雑さについて考察する．R10000 を選択した理由は，以下のとおりである：

- 動的命令スケジューリングの本質的な処理の複雑さが性能を規定するような方式になっている．

† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto Univ.
†† 京都大学工学部情報工学科
Department of Information Science,
Faculty of Engineering, Kyoto Univ.

- 次章で述べる dual-flow の実装との類似点が多く，比較が容易である．

2.1 R10000 の動的命令スケジューリング方式

R10000 は，レジスタ・リネーミングによって out-of-order 実行行う 4-way SS である．図 1 に，R10000 のパイプラインの構成を示す．4 つ同時にフェッチされ(Fetch)，デコードされ(Dec)た命令は，整数/浮動小数点数各 32 エントリの論理レジスタから各 64 エントリの物理レジスタへのリネーミング処理が施される(Rename)．具体的には，命令のターゲットである論理レジスタを物理レジスタの 1 つに割り当て，同時にソースが割り当てられている物理レジスタの番号を得る．各命令は，ここで得られたリネーミング情報とともに整数，ロード/ストア，浮動小数点各 16 エントリの命令キューに格納される．それぞれの命令キュー内で，6b の物理レジスタの番号をタグとするオペランドの待ち合わせ(Cmp)を行う．必要なオペランドが揃った命令の中から，実行ユニットの空きを考慮して実際に実行する命令を選択し(Sel)，実行する．

リザベーション・ステーションを用いる SS では，各実行ユニットは実行結果をリザベーション・ステーションの各エントリに放送しなければならない．そのため，実行ステージのレイテンシが増加し，全体のサイクル・タイムを制約するとされている[2)]．実行ステージのレイテンシは隠蔽することが困難で，サイクル・タイムに直接的に影響するので，深刻である．

R10000 の動的命令スケジューリング方式は，リザベーション・ステーションのデータ系と制御系を，それぞれ物理レジスタ・ファイルと命令キューに分離した方式とすることができる．R10000 の方式では，各実行ユニットは演算結果を物理レジスタ・ファイルとオペランド・バイパスに送るだけでよく，データ・パスは同等の実行ユニット構成を持つ VLIW と同程度に簡単化される．その結果，R10000 の実行ステージはサイクル・タイムを制限しない．

2.2 動的命令スケジューリングの複雑さ

R10000 では実行ステージのレイテンシは VLIW と同程度になため，VLIW に対して速度上不利な点は，動的命令スケジューリングの本質的なオーバーヘッドにあるといえる．そのオーバーヘッドは，主に以下の 3 種の処理からなる．R10000 は，それぞれに半サイクルづつを割り当てている：

(1) **リネーミング**　基本的にはリネーミング情報の格納されたテーブルを読み出すことにより実現するが，更に，同時に処理される 4 命令間の依存を解析する必要がある．24 個の 5b 一致比較器のマトリクスの出力によって，このテーブルに当該サイクルで書き込まれる値をバイパスする．

Fetch | Dec | Rename | Cmp | Sel | Issue | RF → | EX

図 1　R10000 のパイプライン構成（命令フェッチ〜デコード）

(2) **タグ一致検出**　タグである 6b 物理レジスタ番号を最大 80 個の一致比較器に放送しなければならない．

(3) **命令選択**　命令キュー内で発行する命令を選択する．整数命令キューでは，2 つの非均質な ALU に対して，それぞれの ALU で実行可能な命令を 16 エントリの中から 1 つづつ選択しなければならない．簡単のため，命令の古さではなく，キュー内での位置によって発行優先順位を決定している．

これらのうち，リネーミングのレイテンシは，デコード・ステージに複数のサイクルを割り当てることで，分岐予測ミス・ペナルティに転化できる．ただしその結果，VLIW では通常 1〜2 サイクル程度である分岐予測ミス・ペナルティは，R10000 では 4 サイクルにもなっている．

一方，タグ一致検出と命令選択には，本質的に複数のステージを割り当てることができない．発行する命令を決定してから，次に発行する命令を決定するまでを 1 サイクルで実行しないと，実行のレイテンシが 1 サイクルである命令のバイパスが実現できないためである．R10000 では，この 2 つの処理のレイテンシの合計が，サイクル・タイムを制約している．

さて，これら 3 種の処理のうち，命令選択は動的命令スケジューリングにとって本質的で避けることができない．しかし，リネーミングとタグ一致検出はレジスタ間の依存解析を動的に行う処理であるので，何らかの手法で実行前に済ますことができる可能性がある．

3. Dual-Flow

本章では，dual-flow について詳述する．3.1 節ではまず dual-flow の実行モデルについて述べる．Dual-flow では，動的命令スケジューリングのハードウェアを簡略化する代償として実行命令数は SS より増加する．3.2 節では先にその点についてまとめ，SS に対する速度上の優位性は 3.3 節で実装方法を説明する上で明らかにする．なお，コンテクスト，トラップ，投機については，3.4 節でまとめる．

3.1 実行モデル

計算機の実行モデルは，制御駆動型とデータ駆動型に大別される．Dual-flow は，その両側面を持つ：

制御駆動的側面　制御駆動実行モデルとほとんど同様に『制御の流れ』を持つ．プログラムは基本的にメモリに格納された順序で実行され，制御移譲命令によって制御の流れを移譲する．

データ駆動的側面　制御駆動のようなレジスタを定義しない．データ駆動実行モデルのように，データは生産側の命令から消費側の命令により明示的/直接的に受け渡される．

以下では，まず Dual-Flow 実行モデルをより詳細に述べた後，例をあげて実際の動作を説明する．

Dual-Flow 実行モデル

まず，**実行のストリーム**と呼ぶ仮想的なデータ構造を導入する．ストリームは，**スロット**の系列からなる．スロットは 1 回の命令の実行に対応するもので，命令と左右のソース・オペランドを格納するフィールドからなる．実行に先立って空のスロットからなるストリームが無限の未来まで既に存在しているものとする．

ストリームは制御駆動の実行トレースに似ている．命令は，制御の流れにしたがってメモリからフェッチされ，ストリーム上の各スロットの命令フィールドに順に格納される．

各命令は，データ駆動のように，実行結果を命令中に示される宛先に送る．ただし，宛先の指定の方法はデータ駆動実行モデルとは異なる．最も基本的なデータ駆動モデルでは，実行結果の宛先は命令であり，生産側の命令は消費側の命令のアドレスを指示する．Dual-flow では，実行結果の宛先は命令ではなくスロットであり，命令は自命令と宛先スロット間の変位を指定する．

Dual-flow の命令フォーマットを図 2 に示す．命令は 32b 固定長である．宛先を示す d1/2 フィールドは，それぞれ 5b とする．実行結果を送ることができるのは 31 離れたスロットまでに制限される．また s フィールドによって，渡された実行結果が消費側の命令の左右どちらのオペランドになるかも指定する．

スロットのうち，命令と必要なオペランドが揃ったものが，プログラムに記述された順序とは無関係に実行可能となる．以降では,『スロットが，命令とオペランドに対して実行される』という言い方をすることにする．

なお，データ駆動的な振る舞いを導入するのはプロセッサ内部のみであり，プロセッサ外部とのインターフェイスは，制御駆動的と同様，ロード/ストア命令によるメモリ・アクセスによるものとする．Dual-flow は，制御駆動同様のプログラム・オーダを持つので，同じく同様のメモリ・モデルを規定することができる．

実　行　例

図 3 に示す |a − b| を計算するプログラムを例に，dual-flow の動作を説明しよう．このプログラムは，3 行の sub 命令で d = a − b を求め，d が負である場合には，NEG に分岐し，更に 0 − d を計算して最終的な結果とする．プログラムは以下のように実行される．そのストリームを図 4 に示す：

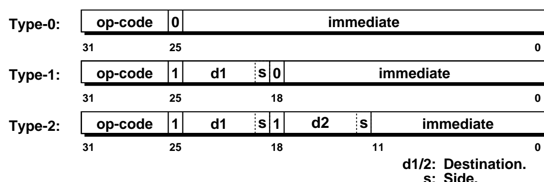

図 2　命令フォーマット

line	label	instruction	
1	A:	imm a	2L
2	B:	imm b	1R
3		sub	1L, 2L
4		bneg NEG	
5	POS:	mov	2L
6		b END	
7	NEG:	rsub 0	1L
8	END:	mov	X

図 3　アセンブリ・プログラム

slot	instruction		OpL	OpR
1	**imm a**	**2L**	**-**	**-**
2	**imm b**	**1R**	**-**	**-**
3	**sub**	**1L, 2L**	**a**	**b**
4	**bneg NEG**		**a**	**-**
5	rsub 0	1L	**d**	-
6	mov	X	-d	-
..	......	......	..	..

図 4　ストリーム

(1) 最初 PC は 1 行を指している．この時点で 4 行の条件分岐命令 bneg までの制御の流れは確定しているので，1～4 行の各命令を，スロット 1～4 の命令フィールドに格納することができる．

(2) 1/2 行の imm 命令は即値を生成する命令で，オペランドを必要としない．したがって直ちに実行されて，値 a/b を 2L/1R で示されるスロット 3 の左/右オペランド・フィールドに格納する．

(3) スロット 3 の sub 命令は，1/2 行の imm 命令からのオペランドの到着によって発火し，演算結果 d は 1L/2L で示されるスロット 4/5 の左オペランド・フィールドに書き込まれる．
1L で示されるスロット 4 の命令フィールドには，既に 4 行の条件分岐 bneg が格納されている．
2L で示されるスロット 5 に入る命令は bneg の結果に依存するので，sub 命令の実行時点では到着していない．スロット 5 は，命令の到着以前に，オペランド d を受け取ることになる．なお，図 4 のボールド・フォントの部分はこの状態での，ローマン・フォントの部分はこれから先の状態を表している．

(4) スロット 5 の命令フィールドには，bneg が not taken であれば 5 行の mov が，taken である場合には 7 行の rsub が格納される．
以降は taken であった場合について説明する．この時点でスロット 5 以降の制御の流れは確定する．

(5) rsub は，sub とは逆に，右オペランドから左オペランドを減ずる命令である．この場合は即値 0 を持っているので，0 − d を計算する．rsub がスロット 5 の命令フィールドに格納される時点では，オペランド d は既に到着しているので，このスロットは直ちに実行される．

(6) 結果−d はスロット 7 の mov 命令によって X に送られる．

Dual-flow は，データ駆動的側面を活かした，条件によってデータを選択する命令を用意する．上記の例では説明のために分岐を用いたが，実際にはこのような場面ではデータ選択命令を用いて分岐命令を削除することが望ましい．

制御駆動では命令が，データ駆動ではデータが，それぞれ計算の主体であると言われる．そのような観点から言えば，dual-flow 実行モデルでは，命令とデータのどちらかがより優位であるということはない．

3.2　ソフトウェアの考慮点

前述の例からの分かるように，Dual-flow のプログラムは，基本的には，制御駆動とよく似ている．ループや関数などは，以下のようにすればよい：

ループ　ほぼ制御駆動と同様に記述できるが，ループ不変数を次のイタレーションに渡すための `mov` 命令が余分に必要になる．

関数　関数への引数は，関数の先頭のスロットから順に，左オペランドとして渡すものと約束する．返り値も同様．なお，戻り先番地とスタック・ポインタも明示的に渡す必要がある．

コード生成に際しては以下の点に注意が必要である：

- 3 個以上のスロット，あるいは，32 以上離れたスロットにデータを送る場合には，`mov` 命令を介する必要がある．
- データの受け渡しが条件分岐をまたぐ場合，データを受け取るスロットの位置，および，オペランドの左右を，taken 側と not taken 側で同一にしなければならない．
- データの受け渡しが関数呼び出しやループをまたぐ場合には，スロット間の距離を静的に決められないので，メモリを介して受け渡す必要がある．

これらの点が性能に与える影響については，4 章で改めて述べる．

3.3　実　　装

本節では，dual-flow の基本的な実装方法について説明し，前節で述べた実行モデルが実装におよぼす効果を明らかにする．

3.3.1　待ち合わせ記憶

Dual-flow の基本的な実装は，R10000 のそれとよく似ている．最も重要な違いは，命令キューと物理レジスタ・ファイルの代わりに，**待ち合わせ記憶**と呼ぶ構造を持つことである．

待ち合わせ記憶は，動的命令スケジューリングにおける中心的役割を果たす．待ち合わせ記憶は，ストリームに対するウィンドウとして動作する．図 4 で示したストリームの振る舞いが，そのまま待ち合わせ記憶上で実現されると考えて差し支えない．なお，実行を完了したスロットは待ち合わせ記憶から順次削除され，解放されたエントリはサイクリックに再利用される．

2章では，SS の動的命令スケジューリングにおける 3 つの処理のうち，命令選択だけが本質的であり，残りの 2 つは無駄であると述べた．Dual-flow では実際に，以下のようにしてその 2 つ ——— リネーミングとタグ一致検出を省略できる：

- ストリーム上のスロットは，待ち合わせ記憶の各エントリに連続的に割り当てられるので，フェッチされた命令は連続するエントリに格納すればよい．
- 待ち合わせ記憶内エントリ番号に 5b の `d1/2` フィールドの値を単に加算することによって実行結果の宛先となるエントリを求められるので，連想アクセスではなく，直接アクセスによってオペランドを受け渡すことができる．

以下では残された命令選択の高速化に主眼を置いて，待ち合わせ記憶の具体的な構成について説明する．

3.3.2　待ち合わせ記憶の構成

待ち合わせ記憶のエントリ数は実装依存である．ただし，最小エントリ数は，命令フォーマット中の `d1/2` のサイズ (5b) から，32 と定められる．待ち合わせ記憶のエントリはサイクリックに再利用されるので，格納先のスロットがまだ使用中である場合がある．その場合には命令のフェッチ/実行を停止する必要があるため，待ち合わせ記憶のエントリ数が小さいと発行可能な命令が減少する．後述するように命令ウィンドウのサイズと待ち合わせ記憶のサイズは独立であり，エントリ数の増加は out-of-order 実行機構の複雑化にはつながらない．したがって，待ち合わせ記憶に対するアクセス・タイムが全体のサイクル・タイムに影響しない範囲で，できるだけ大きくとることが望ましい．

待ち合わせ記憶は，物理的には各フィールドごとに別個のメモリからなり，全体で 1 つの論理的な待ち合わせ記憶として動作する．更に各メモリは，それぞれ複製を用意することによりアクセス・ポート数を削減できる．図 5 に，待ち合わせ記憶の構成例を示す．図では，命令用メモリは実行ユニットごと，左右のオペランド用メモリは，整数系/浮動小数点系ごとに，それぞれ複製している．

オペランド・メモリ

Dual-flow では，1 つの命令が最大 2 個の演算結果を待ち合わせ記憶に書き込むため，オペランド・メモリの書き込みポート数は，同一の演算ユニット構成を持つ SS/VLIW のレジスタ・ファイルの書き込みポー

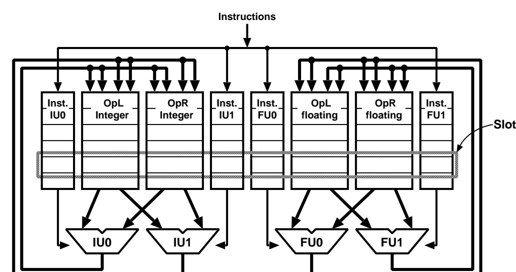

図 5　待ち合わせ記憶の構成

ト数の 2 倍必要になる．

この問題は，左/右オペランド用にそれぞれ別個のメモリを用意することである程度軽減できる．この場合，各メモリの書き込みポート数は SS/VLIW のレジスタ・ファイルのそれの 2 倍必要であることに変わりはないが，読み出しポート数は半分にでき，結果，ポートの総数は同じになる．

あるいは，4 章で述べるように，宛先が 2 つ必要になる命令の数は全体の 20〜30%程度であるので，d2 で示される宛先に対する書き込みをバッファリングすることも考えられる．

ディレクトリ

命令とオペランドの有効性の管理はディレクトリで行われる．ディレクトリはまた，実際に命令選択を担当する部分でもある．

図 6 にディレクトリの基本的な構成を示す．ディレクトリは，基本的には各実行ユニットごとに複製し，優先順位比較器の簡略化を図る．命令，左右オペランドの Ready フィールドは，対応する命令，左右オペランドが到着する時にセットされる．Free フィールドは，当該スロットの実行が決定される時，および，フェッチした命令が当該ディレクトリに対応する実行ユニットに対する命令ではなかった場合にセットされる．

ディレクトリのメモリ部分はシフト・レジスタで構成する．先頭から Free フィールドがセットされているエントリの分だけ全体をシフトしていくことにより，位置情報のみに基づく優先順位比較器によって，最も古いスロットから順に選択することができる．

優先順位比較器には，必ずしもすべてのエントリからの実行要求を入力する必要はない．ただし，優先順位比較器への入力の数が，いわゆる命令ウィンドウのサイズを与えることになる．

3.3.3 パイプライン構成

図 7 に，ALU 命令（実行レイテンシ 1）のパイプラインの構成を示す．実行する命令は OR の直前で決定する必要がある．したがって，図中 A′ 点において，B′ 点までに生成されるオペランドの有効性をほぼ決定できる．しかし，A′ 点は Fetch の途中にあたり，この時点では命令の有効性は分からない．この観点からは，A′ 点の直後では，B′ 点からの実行ステージで実行可能なスロットは，

(1) 実行可能であることが判明しているスロット と，

(2) 命令の到着によって実行可能になるスロット

に分類できる．前者の方がより古いので，まずそれらを優先的に実行することは正しい．しかし，それらを実行した上でなお実行ユニットが空いている場合に，後者のスロットを実行できるようにすることは，分岐予測ミス・ペナルティを削減する上で重要である．

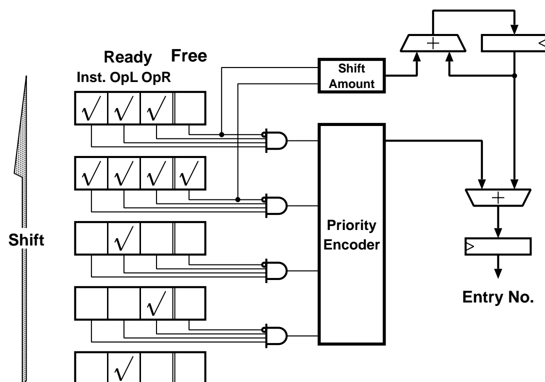

図 6　ディレクトリの基本構成

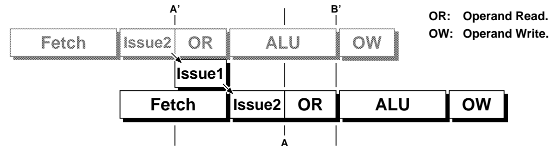

図 7　パイプライン（ALU 命令）の構成

前者のスロットに関しては，命令選択に A′ から A までの 1 サイクルをかけることができるので，余裕をもって実行することができる．後者のスロットに関しても，前述のディレクトリの構成を工夫して，以下の 2 フェーズに分離することができる：

Issue1　既に待ち合わせ記憶中に発行可能な状態で存在しているスロットを調べ，後者のスロットを実行できる実行ユニットの空き状況を確認しておく．

Issue2　Issue1 で調べておいた空いている実行ユニットに対して，後者のスロットの中から実行するものを決定する．

Issue2 の処理の複雑さは，命令ウィンドウのサイズではなく，フェッチ幅の関数になる．命令をキャッシュする時にプリデコードを施せば，処理は更に容易になる．

2章で挙げた SS の動的命令スケジューリングにおける 3 つの処理のうち，リネーミングとタグ一致検出は dual-flow では不要であると述べた．残る命令選択に関しても，上述のように高速化が可能である．Dual-flow の速度は，結局，以下のようにまとめられる：

分岐予測ミス・ペナルティ　リネーミングのため，SS ではデコード・ステージを 1 サイクルとすることは困難である．

Dual-flow では，リネーミングは不要である．更に，命令選択の工夫によって，デコード・ステージを 1 サイクルで済ますことができる．ペナルティは，2 サイクルにおさえられる．

サイクル・タイム　SS では，タグ一致検出から命令選択までを 1 サイクルで実行する必要があり，このレイテンシがサイクル・タイムを制約する．

一方 dual-flow では，命令選択のみをほぼ 1 サイクルかけて実行すればよく，このレイテンシがサイクル・タイムを制約する可能性は低い．

2 サイクルの分岐予測ミス・ペナルティは，VLIW と

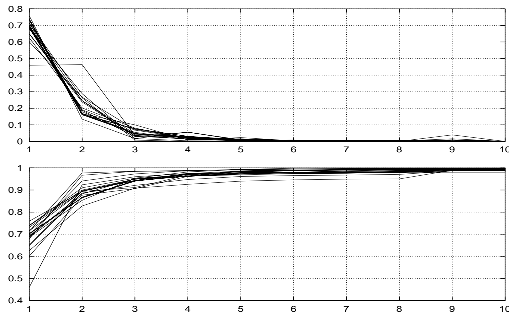

図 8　参照回数

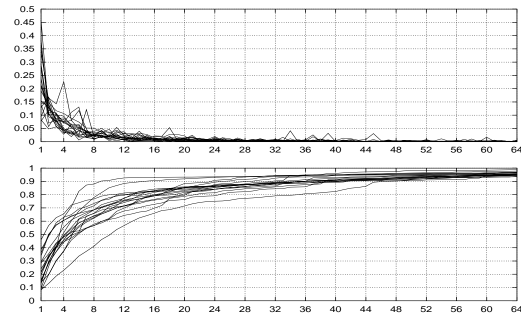

図 9　存続期間

同程度である．また，命令選択ではなく実行ステージのレイテンシがサイクル・タイムを制約する場合には，そのサイクル・タイムは同等の実行ユニット構成を持つ VLIW と同程度となる．dual-flow の速度は，以上の点において VLIW と同程度であり，動的命令スケジューリングの効果の分だけ VLIW より高速化される可能性が高いと結論づけられる．

3.4　コンテクスト，トラップ，投機

Dual-flow アーキテクチャでは，逐次的なマシン状態が定義されない．PC と待ち合わせ記憶内のオペランド・フィールドのイメージがコンテクストを形成する．

制御駆動のように分岐の拡張としてトラップを実現することは不可能である．コンテクストの待避/復帰はハードウェアで行う必要があり，トラップ・ハンドラへの制御の移譲自体が，コンテクストの切り替えを伴う．何らかのハードウェア・サポートが必要であろう．

逐次的なマシン状態を持たないことには一方で，以下のような利点がある．まず，トラップの厳密性について考える必要がない．また，投機との親和性がよくなる．投機的に実行されたスロットは削除せずに残し，ミス時には依存する命令/オペランドを選択的に無効化するだけでよい[3]．依存情報の管理などは必要ない．

4. 予 備 評 価

Dual-flow は，明らかに一時的なデータに対して極端に最適化されているので，一時的ではない ——— 参照回数が多く存続期間が長いデータが多いと，データのコピーを行う命令 (mov) や，ロード/ストア命令の増加に起因する性能の悪化を招く畏れがある．問題になりそうな場面としては，3.2 で述べたループや関数呼び出しといった動的な構造が挙げられる．

SPARC プロセッサで SPEC CPU95 の各プログラムを実行したトレース・データから，参照回数と存続期間を求めた．なお存続期間とは，定義からその最後の参照までの実行トレース上の命令数とする．

結果を図 8 と図 9 に示す．一本の線は，1 つのプログラムに対応している．各図中の上のグラフは分布を，下のグラフはその累積を表している．

参照回数が 3 回以上，存続期間が 32 以上であるようなデータはすべてのデータのうちの，平均で 10%程度，悪い場合でも 20%程度である．これらのデータのために実行される mov 命令の増加も同程度であると予想される．このような目的で挿入される mov 命令はクリティカル・パス上にはないので，その程度の増加であれば，命令発行多重度の向上によって隠蔽できる可能性が高いと思われる．

5. お わ り に

制御駆動とデータ駆動を融合したプロセッサ・アーキテクチャdual-flow について述べた．Dual-flow は，VLIW を越える高速性を達成できる可能性が高い．

Dual-flow は，一時的ではないデータの扱いに弱点があるが，実際のプログラムではそのようなデータは多くはないことが分かった．しかし，この結果から全体的な性能を予測することは困難であり，より厳密な評価が必要である．現在 dual-flow のためのコンパイラを作成中であり，今後はシミュレーションなどによる評価を行う予定である．また，実際にゲート・レベルの設計を行うなどして，VLIW 並のサイクル・タイムが実現できることを検証する必要があるだろう．

謝辞

メンター・グラフィックス・ジャパン株式会社には，Higher Education Program の一環として製品とサービスをご提供いただいたので，ここに感謝する．

本研究の一部は文部省科学研究費補助金（基盤研究(C) #09680334，奨励研究(A) #09780268）による．

参 考 文 献

1) C. Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, No. 4, pp. 28–40 (1996).

2) Hara, T., Ando, H., Nakanishi, C. and Nakaya, M.: Performance Comparison of ILP Machines with Cycle Time Evaluation, *ISCA '23*, Vol. 24, No. 2, pp. 213–224 (1996).

3) 佐藤寿倫: アドレス名前替えによるロード命令の投機的実行, *JSPP '98*, pp. pp. 15–22 (1998).