Vol. 42 No. SIG 9(HPS 3) 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム Aug. 2001

スーパースケーラのための高速な動的命令スケジューリング方式

五島正裕[†] 西野賢悟[†] グェン ハイハー[†]
縣亮慶[†] 中島康彦[†] 森眞一郎[†]
北村俊明[†] 富田眞治[†]

スーパースケーラは，動的命令スケジューリングのため，命令の実行に必要なデータの有効性を追跡する *wakeup* と呼ぶロジックを持つ．従来の *wakeup* は，データに割り当てられたタグによる連想処理に基づくもので，RAM を読み出した結果で CAM をアクセスするという構造を持ち，LSI の微細化，パイプラインの深化にともなっていっそうクリティカルになっていくと予測されている．本稿では *wakeup* を高速化する方式について述べる．本方式は，タグに基づく連想処理ではなく，命令間の依存関係を直接的に表現するテーブルを用いるもので，単に RAM を読み出すことで *wakeup* を実現することができる．さらに本稿では，このロジックの遅延を IPC に対するペナルティに転化する手法を示す．実在する 0.18 μm CMOS プロセスのデザイン・ルールに基づいてこれらのロジックを設計し，回路の面積を求め，Hspice によって遅延を測定した．また，シミュレーションによって，ペナルティを測定した．その結果，3%以下のペナルティを代償に，2 GHz を超える最高動作周波数を達成できることが分かった．

A High-Speed Dynamic Instruction Scheduling Scheme for Superscalars

Masahiro Goshima,[†] Kengo Nishino,[†] Nguyen Hai Ha,[†]
Akiyoshi Agata,[†] Yasuhiko Nakashima,[†] Shin-ichiro Mori,[†]
Toshiaki Kitamura[†] and Shinji Tomita[†]

A superscalar has *wakeup* logic, which manages availability of the data for dynamic instruction scheduling. The usual *wakeup* logic is based on association of the tag allocated to the data. The delay time of *wakeup* consists of read delay time of a RAM and match access delay time of a CAM. Since the delays of these memories are dominated by the wire delay, it will be more critical with smaller feature sizes and deeper pipelines. This paper describes a high-speed dynamic instruction scheduling scheme. This scheme is not based on association of the tags, but a matrix which directly represents the dependence among instructions. The scheme realizes *wakeup* by just reading a small RAM. In addition, this paper also describes a scheme which changes the delay of the logic into IPC penalty. We actually designed the logic guided by a design rule of a real 0.18 μm CMOS process, measured the areas, and calculated the delays by Hspcie. And we also evaluated the penalty of the scheme by simulation. The evaluation result shows that this scheme achieves over 2 GHz clock speed with the IPC penalty less than 3%.

1. はじめに

スーパースケーラの IPC（instructions per cycle）を向上させる最も直接的な方法は，命令発行幅（***IW***: Issue Width）とウィンドウ・サイズ（***WS***: Window Size）を増やすことである．実際初期のスーパースケーラは，トランジスタ数が許す範囲で *IW*, *WS* を増やすことにより，大幅に IPC を向上させてきた．

しかし現在では，LSI の微細化にともなって，トランジスタ数ではなく，クロック速度が *IW*, *WS* を制限する主因となりつつある．*IW*, *WS* を増やしても単純に IPC が向上するわけではないので，いたずらに増加させれば，かえって全体の性能を悪化させることになる．

スーパースケーラの構成要素のうち，***wakeup*** と呼ぶロジックの遅延が，将来クロック速度を制限するものの 1 つになると予測されている[1)]．*wakeup* は，動的命令スケジューリングのために，命令の発行に必要なデータの有効性を追跡するロジックである．

従来の *wakeup* は，各命令が使用するデータに割り当てられた**タグ**による連想処理に基づくもので，RAM

† 京都大学
Kyoto University

を読み出した結果で CAM をアクセスするという構造を持つ．これらのメモリは，配線遅延に支配されるため，LSI の微細化の恩恵を受けにくい．また *wakeup* は，他の多くの構成要素とは異なり，複数のパイプライン・ステージに分割することができない．以上の理由により *wakeup* の遅延は，*IW*，*WS* の増大，LSI の微細化，パイプラインの深化にともなって，スーパースケーラのクロック・スピードを制限する主要因の 1 つとなると予測されるのである．

このような背景から我々は，*wakeup* を高速化するまったく新しいスケジューリング方式を提案する．本方式では，タグによる連想処理ではなく，命令間の依存関係を直接的に表現する行列を用いて *wakeup* を実現する．その結果，4b×4word，1-read *IW*-write という，小型の RAM を読み出す程度の遅延で *wakeup* を実行することができる．本稿では，この方式をさらに高速化する手法と，定量評価の結果を示す．

以下，まず 2 章では従来の動的命令スケジューリング方式の一般的な構成法を紹介し，3 章で提案する方式について詳しく述べる．そして 4 章で，提案方式の定量的評価を行う．

2. 従来の動的命令スケジューリング方式

スーパースケーラを構成するの基本構造のうち，演算器それ自体以外のほとんどすべての遅延は *IW*，*WS* の増加関数で与えられる．そのような構造には，キャッシュ，命令フェッチ・ロジック，レジスタ・ファイル，オペランド・バイパス，そして，本稿の主眼である動的命令スケジューリングを行うロジックなどがある．

ただしそれらの遅延の増大が，直接システムのクロック速度の低下につながるわけではない．いくつかの処理に対しては，パイプライニングやクラスタリング[2),3)]などの技術によって，1 サイクルに終えなければならない処理の遅延を大幅に短縮できるからである．

これらの技術は，ロジックの遅延を，一部の命令の実行レイテンシや，何らかのペナルティに転化するものである．したがって，これらの技術が有効であるためには，クロック速度の向上に対して IPC の悪化の度合いが十分に小さい必要がある．

しかし動的命令スケジューリングを行うロジックに対しては，このような技術は効果的ではない．本章では，その理由について述べる．以下まず 2.1 節においてスーパースケーラの動的命令スケジューリング の原理についてまとめ，2.2 節でスケジューリングの処理と命令パイプラインの関係について説明する．

2.1 動的命令スケジューリングの原理

Out-of-order スーパースケーラは，論理的なレジスタとは別に，リオーダ・バッファや物理レジスタなど，各命令の実行結果を一時的に保存するバッファを用いる．スーパースケーラにおける動的命令スケジューリングは，これらのバッファによって，局所的にデータ駆動型の計算を行うことと見なすことができる．動的命令スケジューリングとは，端的にいえば，左/右のソース・オペランドに対応するバッファのエントリにデータが揃っている命令を選択することである．したがって，原理的には左/右のデータが使用可能かどうかを表すフラグ rdyL/R のテーブルが存在し，スケジューリングにおける中心的な役割を果たす．

動的命令スケジューリングは，次の 5 つのフェーズ——(1) ***rename***，(2) ***register***，(3) ***wakeup***，(4) ***select***，(5) ***issue*** に分けられる．以下では，ある命令 $\boldsymbol{I_c}$ の実行が開始されるタイミングに着目して，スケジューリングの各処理を説明する．なお，I_c の，左ソースは先行する命令 $\boldsymbol{I_p}$ が生成するデータ，右ソースは即値であるとする．

(1) ***rename*** 出力依存や逆依存を動的に解消することを目的に，論理レジスタ番号を個々のデータに割り当てられる ID—**タグ**に付け替える処理を *rename* と呼ぶ．*rename* は，各命令がフェッチされた後，命令ウィンドウへ格納される前に実行される．

タグは個々のデータに付される ID であるから，バックエンドに同時に存在するデータを一意に特定できるものであれば何でもよい（6 章を参照）．最近では，リオーダ・バッファや物理レジスタ・ファイルなど，命令の投機的実行結果を格納するバッファのエントリ ID をタグとして用いることが多い．したがって，以下ではそのように仮定して議論を進めることにする．

さて，I_c には，バッファの空いているエントリが割り当てられる．このエントリの ID—タグを，特に tagD ということにする．同時に I_c は，左/右のソースの論理レジスタ番号から，依存元の命令 I_p に割り当てられた tagD を得る．tagD と区別して，これらを tagL/R ということにする．なお，tagL/R を求める具体的な方法については 3.3.1 項でまとめる．

(2) ***register*** I_c が命令ウィンドウへ格納される際には，I_c に対応する rdyL/R の初期化が行われる．I_c の右オペランドは即値であるので，rdyR は無条件に “1” に初期化する．一方 rdyL の初期値は，I_p の実行がすでに終了しているかどうかによって決まる．I_p の実行がすでに終了していれば，I_p が生成するデータは利用可能であり，rdyL は “1” に初期化される．

以下では I_p がまだ実行されていない場合を考える．その場合，rdyL は “0” に初期化され，I_c は I_p が実行されるのを待って命令ウィンドウ中で『眠る』．

(3) ***wakeup*** やがて，*select* によって I_p が選択され，その実行が決定されると，その結果がある時刻に利用可能になることが分かる．rdyL は適当なタイミングで “1” にセットされ，I_c は『起こされる』．***wakeup*** は，*select* による I_p の実行の決定に従って I_c の rdyL/R をセットする処理をと定義する．

(4) ***select*** rdyL/R がともにセットされている命令が，演算器に対して発行することができる命令である．rdyL/R がともにセットされている最大 WS 個の命令の中から，そのうちの IW 個を選択し，それらに発行許可を与える処理を ***select*** と呼ぶ．

(5) ***issue*** *select* によって選択された命令は，実際に演算器に対して発行される．***issue*** は，具体的には，命令ウィンドウを構成する RAM から選択された命令の情報を読み出す処理である．

***wakeup* と *select* のループ**

wakeup から *select* へは，rdyL/R を介した依存が存在する．また，I_c に対する *wakeup* が開始できるのは，*select* が I_p の実行を決定してからである．すなわち，*wakeup* と *select* は，緊密なフィードバック・ループを形成している．以下では動的命令スケジューリング・ロジックのクリティカル・パスについて考えるが，そこではこのループが特に重要である．

2.2 従来の *wakeup* ロジック

従来の *wakeup* は，基本的に，タグの連想検索によって行われる．すなわち，発行される I_p の tagD に一致する I_c の tagL/R が連想検索され，一致した tagL/R にに対応する rdyL/R がセットされる．

なお，リオーダ・バッファや物理レジスタ・ファイルなどの命令の実行結果を格納するバッファのエントリ ID をタグとする方式では，tagD は，(1) バッファの書き込みアドレスと，(2) *wakeup* の，2 つの用途に使用されることになる．

***wakeup* と *select* の構成**

図 1 に，従来方式の *wakeup*，*select* ロジックのブロック図を示す．ただし実際には，命令ウィンドウは，同図に示したような単一のロジックとしてではなく，演算器のクラスごとに設けられたリザベーション・ステーションや命令キューとして分散実装されることが多い．このような分散化については，2.4 節でまとめる．

wakeup ロジックは，tagD を格納する上部の RAM と，tagL/R をキー，rdyL/R を内容とする下部の CAM からなる．

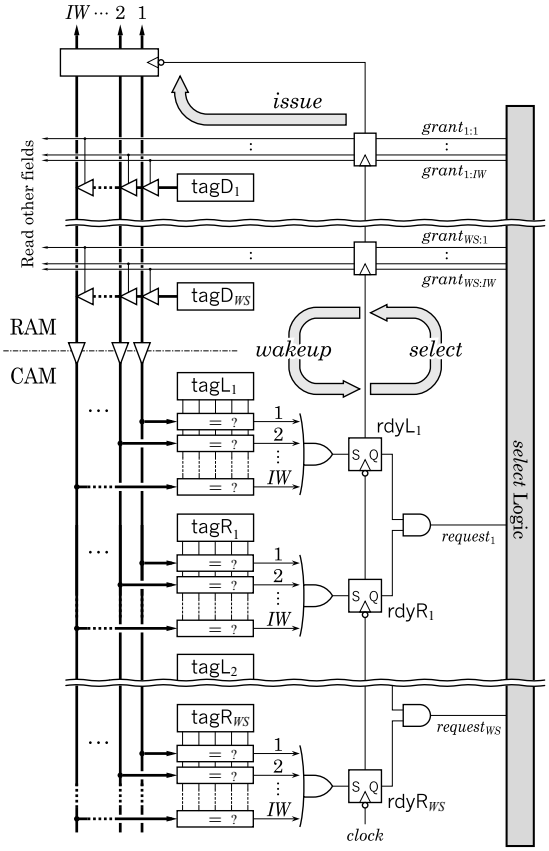

図 1 従来方式の *wakeup* と *select* ロジック
Fig. 1 Usual *wakeup* and *select* logic.

上部の RAM の読み出しワード線には，パイプライン・レジスタを介して，*select* ロジックからの発行許諾信号 *grant* が接続される．したがって，読み出しポートには行デコーダはない．読み出される tagD は，(1) バッファの書き込みアドレスと用いるためには上部のパイプライン・レジスタに，(2) *wakeup* のためには下部の CAM の比較入力ポートに，それぞれ送られる．tagD の読み出しは，したがって，*issue* と *wakeup* の両方に含まれることになる．なお，上部のパイプライン・レジスタに送られた tagD は，適当なサイクル遅延されたうえで，バッファの書き込みアドレスとして使用される．

下部の CAM では，比較入力ポートに入力された tagD と一致する tagL/R が連想検索され，対応する rdyL/R がセットされる．CAM の出力側では，rdyL/R を記憶する RAM セルの出力信号が，組合せ回路的に *select* の入力 *request* に接続されている．

select は，WS 個の *request* から IW 個を選んで，対応する *grant* をアサートする．

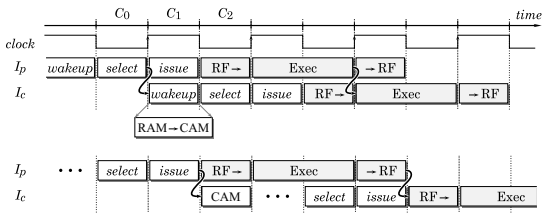

図 2 命令パイプラインにおける *wakeup*, *select*, *issue*
Fig. 2 Pipelining *wakeup*, *select* and *issue*.

wakeup と *select* の動作

図 2 上に，*wakeup*, *select*, *issue* の命令パイプラインでの位置づけとその動作の様子を示す．同図は，MIPS R10000 のパイプライン構成に準ずる[4]．図中，Exec は実行を，RF→ と →RF は物理レジスタ・ファイルに対する読み出しと書き戻しを表す．同図は，タイミングが最もクリティカルな場合，すなわち，レイテンシが 1 である命令 I_p の次のサイクルで I_c が実行される場合を示している．I_p が生成したデータは，オペランド・バイパスを通って I_c の実行に使用される．

ロジックの動作は，以下のようにまとめられる：

$C0$　*select* が I_p を選択し，I_p に対応する *grant* をアサートする．

$C1$　I_p に対する *issue* と同時に，I_c に対する *wakeup* が行われる．*wakeup* では，RAM から読み出された I_p の tagD が CAM に入力される．その結果，I_c の rdyL がセットされ，対応する *request* がアサートされる．

$C2$　この *select* では，I_c も選択の対象となる．ここで実際に I_c が選択されると，今度は I_c に対応する *grant* がアサートされる．

wakeup と *select* のフィードバック・ループは，回路上では以上のように現れる．

2.3 動的命令スケジューリングとパイプライン

では，動的命令スケジューリングの各処理のパイプライン化可能性について考えよう．

スケジューリングの処理のうち，フロントエンドにある *rename* と *register* は，パイプライン化することでクリティカル・パスから外すことができる．実際，現存するスーパースケーラでは，特に *rename* の遅延のため，デコード・ステージに複数サイクルを充てることが普通である．デコードに複数サイクルを費やした場合，そのサイクルは分岐予測ミス・ペナルティの一部に変わる．

また，*issue* は，*rename*, *register* と同様，パイプライン化可能である．図 2 上に示した例では，*issue* に 0.5 サイクルをかけているが，それによって以降のステージが全体に 0.5 サイクル後にずれるだけで，スループットには影響がない．*issue* に費やすサイクルは，やはり分岐予測ミス・ペナルティの一部に変わる．

しかし，*wakeup* と *select* は，実際上パイプライン化することができない．図 1 において，RAM と CAM の間にパイプライン・レジスタを配置した場合を考えよう．あるいは，*issue* で読み出された tagD を CAM に入力すると考えてもよい．その場合のパイプラインの様子を図 2 下に示す．同図から分かるように，データが利用可能になっているにもかかわらず，I_c は I_p の次のサイクルには発行されなくなる．

このことは，ちょうど，レイテンシが 1（サイクル）である演算器からはオペランド・バイパスを行わないことと等価である．詳細は 4 章で述べるが，それによる IPC の悪化は 30%程度にもなり，パイプライン化によるクロック速度の向上に見合わない可能性が高い．レイテンシが 1 である演算器からのオペランド・バイパスが利用できるパスを**レイテンシが 1 のパス**と呼ぶことにしよう．すると，レイテンシが 1 のパスでは，*wakeup* と *select* は合わせて 1 サイクルで実行しなければならないとすることができる．

2.4 命令ウィンドウの分散化

前述したように，命令ウィンドウは，集中化された単一のロジックとして実装されるのではなく，演算器のクラスごとに設けられたリザベーション・ステーションや命令キューとして，分散実装されることが多い．ロジックの遅延を考えるにあたってはこの分散化が重要である．このような分散化は，IPC に対する影響も小さいうえに，遅延を大幅に短縮できるからである．

分散化には，(1) 構成要素のパラメータの縮小と，(2) パスのレイテンシに合わせた最適化，の 2 つの効果がある．以下，それぞれについて説明する．

(1) パラメータの縮小

複数の命令キューに分散化した場合にも，各キューのロジックの構成自体は図 1 に示したものと基本的には変わらない．キューの命令発行幅を $\boldsymbol{IW_q}$，サイズを $\boldsymbol{WS_q}$ とすると，図中の IW, WS を IW_q, WS_q に読み替えればよい．

ただし，CAM の比較入力ポート数だけは，IW から IW_q にすることはできない．RAM から読み出された tagD は，別のキューの CAM にも送る必要がある．逆に，各キューの CAM の比較入力ポート数は，別のキューからの tagD を受け入れるため，IW_q とするのではなく，IW のままとするのが理想である．実際には，IW_q よりやや大きい数に制限し，IPC とのトレードオフとすることが多い．

(2) レイテンシに合わせた最適化

前述したように，レイテンシ 1 のパスでは *wakeup* と *select* は合わせて 1 サイクルで実行する必要がある．逆に，これ以外のパスでは，*wakeup* と *select* を適当にパイプライン化してもよく，クリティカル・パスについて議論からは除外してもよい．

たとえば，MIPS R10000[4]は，整数演算，ロード/ストア（**LS**），浮動小数点（**FP**）演算のそれぞれに命令キューを用意している．レイテンシ 1 のパスは，(1) 整数から整数，(2) 整数から LS，の 2 つである．したがって，クリティカル・パスについての議論では，特に整数キューのみを考慮すればよい．

R10000 の整数キューの *wakeup* と *select* ロジックも，基本的にはこれまで述べてきたとおりである．そのパラメータは，$(IW_q, IW, WS_q, TW) = (2, 4, 16, 6)$ である．TW はタグのビット幅である．ただし，ソース・オペランドは，L/R の 2 つではなく，A，B，C の 3 つある．A，B に対しては，自身から 2 つ，ロード命令から 1 つの，C に対しては，それらに加えて，FP 比較命令から 1 つの tagD を受け入れる．したがって，RAM は 2-read 2-write，6b×16word，CAM のキー部は 2-write，6b×48word である．CAM の比較入力ポート数は 3〜4，比較器の数は 160 個にものぼる．

従来の *wakeup* では，このような RAM と CAM を約 0.5 サイクルの間に逐次的にアクセスしなければならない．詳細は 5 章で述べるが，この程度の IW_q，WS_q でも，その遅延はかなりクリティカルである．

現在，スーパースケーラのパイプラインはますます深くなる傾向にある．*wakeup* と *select* はパイプライン化できないため，パイプラインが深くなると相対的にクリティカルになる．

これらのうち *wakeup* には，LSI の微細化の恩恵を受けにくいという問題もある．*select* の遅延はもっぱらゲート遅延からなるため，微細化にともなって順調にスケーリングされる．一方 *wakeup* の遅延は主に RAM と CAM のワード線，ビット線，マッチ線などの配線の遅延からなるため，スケーリングされにくい．したがって，LSI が微細化されるにつれ，*wakeup* の遅延は相対的にクリティカルになる[1]．

以上の理由から，*wakeup* は，*IW*，*WS* の増大，パイプラインの深化，LSI の微細化にともなって，スーパースケーラのクロック速度を制限する主要因の 1 つとなると考えられるのである．次章以降では述べる提案手法は，この *wakeup* を置き換えるものである．

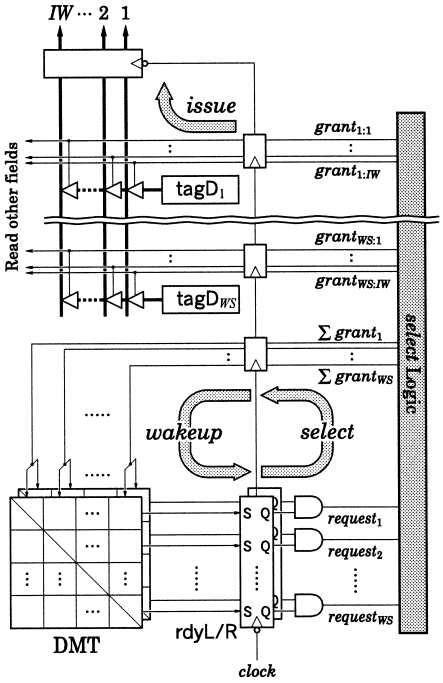

図 3 スケジューリング・ロジックにおける DMT の位置づけ
Fig. 3 Position of DMT in scheduling logic.

3. 提案する動的命令スケジューリング方式

提案するスケジューリング方式は，前章で述べた従来方式とは根本的に異なり，タグを用いずに *wakeup* を行う．提案方式では，命令ウィンドウ中の各命令間のデータ依存関係を直接的に表す**依存行列テーブル**（dependence matrices table: **DMT**）が，スケジューリングにおける中心的な役割を果たす．

3.1 DMT の概要

図 3 に，スケジューリング・ロジックにおける DMT の位置を示す．図 1 に示した従来方式のロジックと比較されたい．提案方式では，従来方式における RAM と，CAM のキー部が，DMT に置き換わる．DMT は，*select* ロジックからの発行許諾信号（*grant*）を受けて rdyL/R の入力を生成する．

提案方式では，*wakeup* にはタグを用いない．タグ☆が命令の実行結果を格納するバッファのエントリ ID である場合には，従来方式と同様タグを読み出す必要があるが，それは *issue* に含まれ，*wakeup* には含まれない（図 3）．前述したように，*issue* は，*wakeup* とは異なり，パイプライン化可能である．

☆ この場合，タグというネーミングはもはや適当ではないが，説明の連続性のためあえてそのままとする．

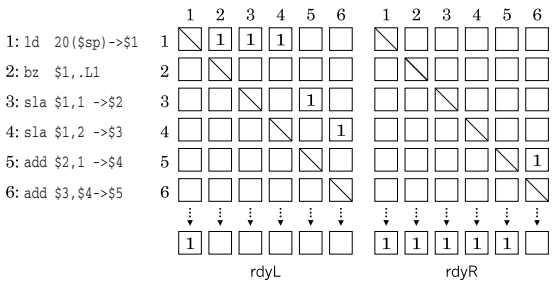

図 4 依存行列テーブル：DMT
Fig. 4 Dependence matrices table: DMT.

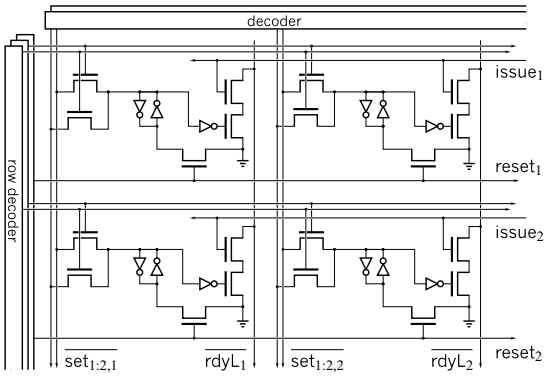

図 5 DMT のロジック（左オペランド用）
Fig. 5 Logic of DMT (for left operand).

DMT の定義

図 4 に，DMT の概念図を示す．ここでいう依存行列は，命令間のデータ依存関係を表す $WS \times WS$ の行列で（対角要素は使用しない），左/右のソース・オペランドに対して 1 つずつ用意される．エントリ ID $= p$ の命令が生産するデータを，ID $= c$ の命令がその左（右）オペランドとして消費する場合，左（右）オペランド用の行列の p 行 c 列の要素は “1”，それ以外は “0” とする．

図 4 に示す例では，連続する 6 つの命令が ID = 1 のエントリから順に格納された場合を表している．この場合，たとえば，ID = 1 の命令が生産する $1 を ID =2，3，4 の命令が左オペランドとして消費するから，左の行列の 1 行目では 2，3，4 列の要素が “1” となる．その他の要素も同様に求められる．

DMT による *wakeup*

wakeup においては，発行される IW 個の命令に対応する IW 行の bitwise-OR を求めれば，セットすべき rdyL/R を表す行ベクトルを求めることができる．

図 4 に示した状態でエントリ ID = 1 の命令が発行された場合を考えよう．DMT の第 1 行は，定義により，ID = 1 の命令が実行されるときにセットすべき rdyL/R を表している．実際に ID = 1 の命令が実行されると，ID = 2，3，4 の rdyL がセットされ，次のサイクルにはその 3 つの命令が実行可能になる．実際にその 3 つの命令が選択され同時に実行される場合には，第 2，3，4 行の bitwise-OR を求めればよい．すると，ID = 5，6 の rdyL をセットすればよいことが分かる．ただし実際には，bitwise-OR を求めるといっても，各行で “1” である要素ははたかだか 1 つである；なぜなら，1 つのソース・オペランドを生産する命令はただ 1 つだからである．

次節からは，DMT のロジックとその動作について詳細に説明する．DMT の動作としては，*wakeup*，更新のほかに，投機の失敗時の状態回復がある．3.2 節ではロジックの構成を示し，*wakeup* 時の動作を説明する．3.3 節では更新時の動作，3.4 節では投機失敗時の状態回復について述べる．

3.2 DMT の実装

前述したように，DMT の読み出しでは，原理的には複数行の bitwise-OR を求める必要がある．しかし，そのための特別なロジックは必要なく，通常の 1-read の RAM をほぼそのまま用いることができる．

図 5 に，DMT の左オペランド用の 2 行 2 列分の回路図を示す．各セルの中央には 4T セルが，その左右には更新用の書き込みポートが配されている．更新に関しては，次節で詳しく述べる．

wakeup に関連するのは，4T セルの右側にある読み出しポートである．図から分かるように，この読み出しポート部は，single-bitline の 1-read の RAM と，基本的には同じものである．

ただし，通常の RAM では同時にはたかだか 1 つのワード線しかアサートされないのに対して，DMT では毎サイクル実行される I_p に対応する（最大）IW 本のワード線 issue が同時にアサートされる．各ビット線 $\overline{\text{rdyL}}$ には，アサートされた issue に対応する（最大）IW 個のセルが接続され，いずれかのセルの出力が low であれば pull-down される．すなわち，単に複数のワード線を同時にアサートすることによって，対応する行の bitwise-OR を読み出すことができる．

なお DMT では，*select* ロジックからの入力信号が若干異なる．従来方式の *wakeup*，および，両方式の *issue* では，ポートを選択するため，*select* ロジックからは各命令が何番目に選ばれたかを示す信号が必要であった．一方 DMT には，何番目に選ばれたかという信号は必要なく，選ばれたかどうかを表す信号が必要である．*select* の構成によっては，前者の OR をとる必要がある場合がある（5.1.4 項参照）．

3.3 DMT の更新

次に，DMT の更新処理について述べる．DMT の更新は，ちょうど *rename* と同時に，*rename* と同形のロジックによって実現される．そこで，以下まず 3.3.1 項で *rename* についてまとめた後，3.3.2 項で DMT の更新について述べる．

3.3.1 *rename*

rename は，前述したように，tagD を割り当てる処理と，対応する tagL/R を求める処理からなる．割り当てるべき tagD は，たとえば直前のサイクルで求めておくことができるから，性能上重要であるのは tagL/R を求める処理である．

tagL/R を求めるには，論理レジスタ番号からタグへの写像を保持する**レジスタ・マップ・テーブル**（**RMT**）が中心的な役割を果たす．RMT は，論理レジスタ番号をアドレス，タグを内容とする，$2{\cdot}IW$-read IW-write の RAM である．

各命令に割り当てられた tagD は，デスティネーションの論理レジスタ番号をアドレスとして RMT に書き込まれる．tagL/R は，基本的には，左/右ソースの論理レジスタ番号をアドレスとして RMT を読み出すことによって得ることができる．

ただし，同時にデコードされる（最大）IW 個の命令間に依存がある場合には RMT からは『古い』tagL/R が得られるので，依存検出器が必要となる．依存検出器は，同時にデコードされる命令間で論理レジスタ番号の比較を行う一致比較器のアレイである．依存が検出された場合には，RMT から読み出される『古い』tagL/R の代わりに，RMT に書き込まれようとしている『新しい』tagD を用いればよい．

3.3.2 DMT の更新

次に，DMT の更新処理について述べる．今デコードされている I_c が ID $= c$ のエントリに格納されるとしよう．DMT の更新処理は，以下の 2 つのフェーズからなる：すなわち，(1)I_c の依存元の命令 I_p が格納されているエントリの ID $= p$ を求める，(2) DMT の p 行 c 列をセットする，の 2 つである．

(1) p は，*rename* とまったく同じ方法によって求めることができる．すなわち，上述の *rename* の処理の説明において,『タグ』を『I_p のエントリ ID』と，『RMT』を『**生産者テーブル**（producer table: **PT**)』と読み換えればよい；ただし PT は，論理アドレス番号をアドレスとし，I_p のエントリ ID を内容とする，$2 \cdot IW$-read IW-write の RAM である．

DMT の更新処理においても，*rename* と同様の依存検出器が必要となるが，これは *rename* 用のものと共用することができる．したがって，提案方式のために別途必要となるのは主に PT である．

(2) では，DMT に p をアドレスとして与え，c をデコードしたものをセットする．この際には，上書きするのではなく，ビット・セットを行わなければならない点に注意する必要がある．エントリ p には，一般に，すでに “1” となっているビットがある．したがって，通常の RAM の構成では，エントリ p を読み出し，c で示されるビットをセットし，その結果を書き戻す read-modify-write の動作が必要となる．

図 5 に示した DMT の回路では，read-modify-write を避けるため，通常の RAM では相補的に用意される書き込みポートの一方をあえて省略してある．同図中，4T セルの左側がセットを行うためのポートである．同図は $IW = 2$ の場合であり，セット用の nMOS トランジスタが各セルに 2 つずつ用意されている．セットを行うときには，c をデコードしさらに反転したものをビット線に入力する．各セルにおいて，接続されたビット線が low であればセットが行われる．ビット線が high であっても，low レベルしか伝達しないという nMOS の性質により，すでにセットされているビットがリセットされることはない．

この回路では，read-modify-write が避けられる代わりに，各エントリは使用に先だってリセットする必要がある．リセット用ポートは，同図では，4T セルの右下に示してある．リセットのため，パイプラインの構成によっては，エントリが解放された直後のサイクルには当該エントリを使用することができなくなる．しかし，4 章で示すように，その影響はわずかである．

3.3.3 更新処理の遅延

次に，更新処理の遅延について考察する．DMT の更新は *rename* と同時に行われる．その遅延がクリティカルとなる可能性があるのは，(1) PT の遅延 と，(2) p のデコードであるが，以下のように，双方とも *rename* に比べて短いと考えられる：

(1) バッファのエントリ数は WS の 1〜2 倍程度とすることが普通である☆から，RMT の内容であるタグは，PT の内容であるエントリ ID より 0〜1 b 程度長い．その他のパラメータは両者で同じであるから，PT の遅延は RMT の遅延と同じかより短い．

(2) DMT への書き込みは，*rename* で得られた tagL/R を含む，デコードされた命令の情報の命令ウィンドウへ書き込みと同時に行われる．この書き込みの

☆ その不足によって命令フェッチが滞ることを防ぐには，バッファは最低 WS エントリ必要である．リネーミングのオーバヘッドのため，それでは不十分な場合がある．

遅延は，通常，ウィンドウを構成する RAM の行デコーダとワード線の遅延によって決まる．p のデコードはこの行デコーダと同じ回路で実現できるから，その遅延が表面化する可能性は低い．
以上から，提案方式における DMT の更新処理がクロック速度を低下させる可能性は低いといえる．

3.4 DMT と投機

最近のスーパースケーラでは，分岐予測による投機的実行が必須となっている．命令スケジューリングの方式にとっては，投機失敗時の処理が容易であることが重要である．

DMT における分岐予測ミス時の処理はごく簡単である．分岐予測ミス時には，rdyL/R を含む，無効化される命令に対応する列をすべて “0” にリセットすればよい．図 4 の例において ID = 2 の分岐命令が予測ミスを起こした場合には，ID = 3〜6 の命令が無効化される．DMT では，3〜6 列をリセットすればよい．

列のリセットは命令ウィンドウの他の情報の無効化と同程度のコストで実行できるから，投機失敗時に DMT に関連して新たなペナルティが生じることはない．

3.5 DMT の高速化

本節では，DMT アクセス，特に *wakeup* のための DMT の読み出しを高速化する方法について述べる．その 1 つ目は，2.4 節で述べたのと同様の命令ウィンドウの分散化である．2 つ目は，DMT を縮小することにより，そのレイテンシを IPC に対するペナルティに転化する方法である．両者はそれぞれ独立に適用することができるが，組み合わせることでさらに効果を発揮する．したがって，以下，まず 3.5.1 項で分散化について述べた後，3.5.2 項では，分散化されたそれをさらに縮小するという流れで説明を行う．

3.5.1 DMT の分散化

命令ウィンドウが，それぞれ発行幅 IW_q，サイズ WS_q の q 本の命令キューに分散化される場合には，DMT は q 個の WS_q 行 WS 列の部分行列に分割される．R10000 は，整数，LS，FP の 3 つの命令キューを持ち，それらのパラメータはそれぞれ $(IW_q, WS_q) = (2, 16)$ である．この場合 DMT は，3 個の 16 行 48 列の部分行列に分割される．その様子を**図 6** に示す．

従来方式と同様に，提案手法でも，分散化によって，(1) パラメータの縮小と，(2) レイテンシに合わせた最適化，の 2 つの効果を得ることができる．

(1) パラメータの縮小

さて，それぞれの部分行列では，書き込みポート数が IW から IW_q に削減されるため，セル面積が減少し，読み出し遅延も短くなる．

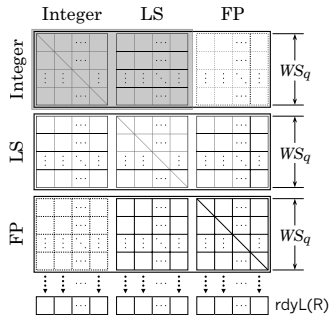

図 6 DMT の分散化
Fig. 6 Distribution of DMT.

(2) レイテンシに合わせた最適化

図中 9 個ある WS_q 行 WS_q 列の小行列は，整数，LS，FP 間のデータの依存を表す．SPARC[6] などのように，整数—FP レジスタ間転送命令を持たないアーキテクチャでは，図中で破線で示した右上と左下の小行列は不要である．また，MIPS などのように転送命令を持つアーキテクチャであっても，その命令の終了まで後続の命令のフェッチ，デコードを中断することによって，省略することができる．その場合 rdyL/R は，*register* において “1” に初期化される（2.1 節参照）．いずれの小行列もこの手法によって省略することができるが，上記以外では IPC に与える影響が大きいので，やめておいた方がよい．

また，前述したように，*wakeup* と *select* を合わせて 1 サイクルで実行する必要があるのはレイテンシが 1 のパスだけであり，R10000 の構成では，整数から整数，整数から LS の 2 カ所だけがこの条件を満たす．したがって，図 6 で影を付けた部分の読み出しは *select* と合わせて 1 サイクルで実行する必要があるが，それ以外の部分の読み出しは適当にパイプライン化してよい．

図 6 で影を付けた部分のみを取り出し，これを L1-DMT と呼ぶ．元の DMT を L2-DMT と呼ぶ．

L1-DMT は，元の DMT に比べて大幅に小型化されるため，読み出しの遅延も短縮される．

L2-DMT の読み出しは，対応する演算器のレイテンシに合わせて，*select* と合わせて 2 サイクル以上かけて実行すればよい．*wakeup* と *select* にそれぞれ 0.5 サイクルを充てるとすると，L2-DMT の読み出しには 1.5 サイクル，すなわち，L1-DMT の読み出しの 3 倍の時間をかけられる．したがって，L2-DMT の読み出しがクリティカルになる可能性は非常に低く，*wakeup* の遅延についての議論から除外することができる．

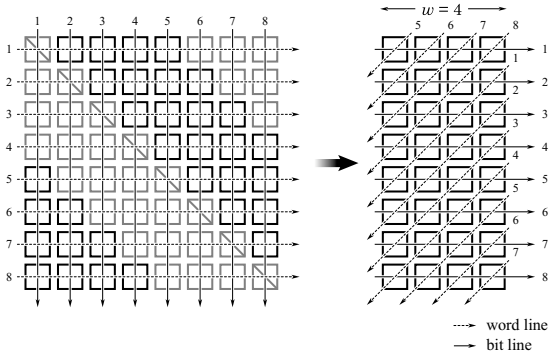

図 7 L1-DMT の縮小
Fig. 7 Shrinking L1-DMT.

さらに，L1-DMT に対しては，以下に述べる高速化手法を適用することができる．

3.5.2 L1-DMT の縮小

依存する命令間の距離は短い場合が多く，32 命令以下の場合が 90%程度以上を占めることが分かっている[7]．この性質を利用して，L1-DMT を縮小し，さらにその遅延を短縮することを考える．すなわち，各命令キューにおいて自命令の後の $\boldsymbol{w}$ $(\mathbf{1} \leq \boldsymbol{w} \leq \boldsymbol{WS_q} - \mathbf{1})$ 個の命令に対応するビットだけを L1-DMT に残し，その他のビットを L2-DMT に移すのである．

図 7 に，$WS_q = 8$，$w = 4$ の場合の L1-DMT の縮小の様子を示す．図には，整数から整数の部分のみを示したが，整数から LS の部分も同様である．左は縮小前の，右が縮小後の L1-DMT である．元の L1-DMT から削除されるセルは，左図中で薄く示した．必要なセルを矩形領域に集めるにはいくつかの方法が考えられるが，よりクリティカルであるビット線 $\overline{\text{rdyL/R}}$ を短縮することを優先して，図 7 右のようにするとよいであろう．w を L1-DMT の**幅**と呼ぶことにする．

図 7 右から分かるように，縮小された L1-DMT のワード線 issue，ビット線 $\overline{\text{rdyL/R}}$ は，それぞれ w 個のセルにしか接続されていない．したがって L1-DMT の読み出し，すなわち，$wakeup$ の遅延は，WS_q とは独立に，w によって決めることができる．

さて，依存する命令間の距離が w 以下の場合には，縮小した L1-DMT によって rdyL/R を更新できる．w を超えていた場合には，rdyL/R は L2-DMT によって更新される．その場合には，更新が 1 サイクル遅れるため，依存先の命令は依存元の命令の次のサイクルには発行することができなくなる．しかし，依存する命令間の距離は短い場合が多いため，次章で示すように，その影響は十分に小さい．

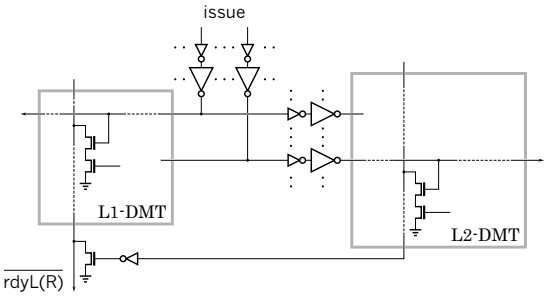

図 8 DMT の 2 階層化
Fig. 8 Two-level DMT.

L1/2-DMT 間の接続

L1-DMT の読み出しでは，L2-DMT の影響を最小化することが望ましい．特に，L1-DMT を縮小した場合には，整数命令に対する発行信号 issue は，L1/2-DMT 双方のワード線を駆動する必要があるので，注意する必要がある．**図 8** に，両者の接続例を示す．なお，同図中には示していないが，L2-DMT のパスの適当な位置にはパイプライン・レジスタが挿入される．同図の構成では，L2-DMT が L1-DMT の読み出しの遅延に与える影響は以下の 2 点になる：

- L2-DMT のワード線ドライバのため，L1-DMT のワード線ドライバの負荷が増加する．
- L2-DMT の読み出し結果で L1-DMT のビット線をプルダウンするため，ビット線の容量が増加する．

いずれも，その影響はごくわずかである．

パディング

縮小した L1-DMT によって rdyL/R が更新できるのは，実際には，命令間の距離ではなく，命令が格納されたウィンドウのエントリ間の距離が w 以下の場合である．したがって，L1-DMT の縮小が有効であるためには，命令流上での距離とエントリ間の距離がある程度一致するように制御する必要がある．

同一のキュー内の命令に対しては，エントリをサイクリックに利用することによって比較的容易にこの条件を満たすことができる．しかし異なるキュー間では，それに加えて，キュー間でのエントリの使用位置の制御が必要になる．R10000 の構成では，LS 命令を，依存元の整数命令が格納されるエントリの『横』以降のエントリに格納する必要がある．

この制御自体は容易であり，それがクロック・スピードやデコードの遅延に悪影響を及ぼす可能性は低いが，LS キュー・エントリの利用効率は悪化してしまう．命令間の依存関係を調べることでこの影響は緩和できるが，整数命令が格納されたエントリより『前』の LS

表 1 プログラム
Table 1 Programs.

プログラム	入力セット	実行命令数
099.go	9 9	132 M
124.m88ksim	dcrand.big	120 M
126.gcc	genrecog.i	122 M
129.compress	10000 q 2131	35 M
130.li	train.lsp	183 M
132.ijpeg	vigo.ppm -GO	26 M
134.perl	primes.in	10 M
147.vortex	persons.250	157 M

表 2 リセットとパディングによる IPC の低下率（%）
() 内はプログラム番号
Table 2 IPC drop rate of reset and/or padding (%).

		min	max	av
x1	reset	0.01 (124)	0.75 (132)	0.25
	padding	0.04 (124)	1.23 (147)	0.47
	both	0.10 (124)	1.75 (147)	0.74
x2	reset	0.01 (134)	0.77 (132)	0.18
	padding	0.02 (124)	1.01 (132)	0.28
	both	0.05 (124)	1.76 (132)	0.48

キュー・エントリを，盲目的にパディングによって埋めてしまっても，次章で示すように，その影響はごくわずかである．

4. IPC の評価

SimpleScalar ツールセット[8]（ver.2.0）に対して，3.5 節で述べた DMT の縮小を実装し，**表 1** に示す SPEC CINT95 の 8 つのプログラムを用いて，DMT の幅に対する IPC の変化を測定した．

ベース・モデルとしては，MIPS R10000[4]のマシン構成を用いた．R10000 は，LS，整数演算，FP 演算のそれぞれに命令キューを持ち，$(IW_q, IW, WS_q, TW) = (2, 4, 16, 6)$ である．ただし，IW_q，WS_q は，各キューの発行幅とサイズ（2.4 節参照），TW はタグのビット幅である．1 次キャッシュは，命令/データ，それぞれ，容量 32 KB，ライン・サイズ 32 B である．2 次キャッシュは命令/データ共有で，容量 1 MB，ライン・サイズ 64 B である．レイテンシは，1 次は 1 サイクル，2 次は 6 サイクルである．2 次キャッシュ・ミス時は，最初のデータが得られるまでが 18 サイクルで，後続データへのアクセスには 2 サイクルが必要である．分岐予測には，ツールセットに用意されている 2 b 飽和型カウンタによるもの（bimod）を用いた．これを，構成 R10K×1 と呼ぶことにする．また，キャッシュ– メモリ以外のすべての資源を 2 倍，すなわち，$(IW_q, IW, WS_q, TW) = (4, 8, 32, 7)$ としたものもあわせて測定した．これを構成 R10K×2 と呼ぶ．

以下まず，3.3.2 項で述べた DMT エントリのリセットと，3.5.2 項で述べたパディングの影響を調べ，その後に DMT の縮小について述べる．

リセットとパディングの影響

表 2 に，構成 R10K×1 の従来方式に対して，リセット，パディング，および，その両方を行った場合の IPC の低下率を示す．これらによる IPC の低下率は 1%程度以下である．後述する DMT の縮小による IPC の低下は数%程度はあるから，リセットとパディングによる影響はそれに比べてほとんど無視できる．

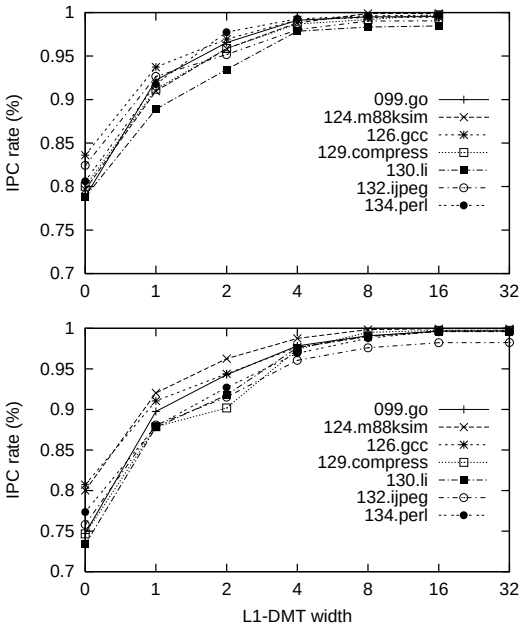

図 9 DMT の幅に対する IPC
Fig. 9 IPC rate versus DMT width.

L1-DMT の縮小の影響

次に，L1-DMT の縮小を行った場合の，従来方式に対する IPC の変化を**図 9** に示す．グラフの横軸は DMT の幅，縦軸は従来方式に対する IPC の百分率である．1 本の曲線は，1 つのプログラムに対応する．なお，$w = WS_q$，すなわち，L1-DMT の縮小を行わない場合には，パディングを行う必要はないが，同グラフではあえて行った場合の値を示している．$w = WS_q$ の場合の値が 100%より小さいのは，リセットとパディングの影響による．

同グラフからは，L1-DMT の幅は，WS_q の 1/4 あれば，従来方式に対する IPC の低下は 3%以下に抑えられることが分かる．

ちなみに，$w = 0$ のときの値は，ちょうど，2.3 節で述べた，*wakeup* と *select* にそれぞれ 1 サイクル

をかけた場合に相当する．その場合の IPC は低下は，30%程度にもなる．

5. 回路の評価

富士通株式会社から提供された，CS80A プロセスのデザイン・ルールに基づいて，整数命令キューにおける従来方式と提案方式の主要な回路のレイアウト設計をスクラッチから行った．得られたレイアウトから，回路面積を求め，また，プロセス・パラメータに基づいて RC データを抽出し，Hspice シミュレーションによって遅延を求めた．

CS80A は，ゲート長 0.18 μm のバルク CMOS プロセスで，ゲート絶縁膜，層間絶縁膜は SiO_2 である．メタルは 6 層アルミであるが，評価では 3 層のみを用いている．

各回路の評価でも，IPC の評価でも用いたのと同じ，構成 R10K×1 と R10K×2 を対象とした．それぞれの (IW_q, IW, WS_q, TW) は，前述のとおり，$(2, 4, 16, 6)$，および $(4, 8, 32, 7)$ である．DMT に対しては，幅が 1，2，4，8，16，32 の 6 種を評価した．

以下，5.1 節では設計した回路の説明を行い，5.2 節で評価結果を示す．

5.1 回路の説明

以下，順に，従来方式の tagD 用 RAM と CAM，次いで DMT，最後に *select* の説明を行う．

5.1.1 RAM

従来方式における tagD 用 RAM は，IW_q-read IW_q-write，TW b×WS_q word である．

図 10 上に，RAM セルの回路図を示す．同図は，$IW_q = 2$ のもので，2-read 2-write となっている．中央に 4T セル，その左右上部に書き込みポート，下部にインバータを介して読み出しポートを配置している．

RAM を設計するにあたっては，読み出しの遅延に関して，セル・サイズをできるだけ小さくする方針と，大きくする方針がある．前者は，セルをできる限り小さくし，その結果ゆっくりと変化するビット線の電位をセンス・アンプを用いて読み出す方針である．後者は，セル・サイズは犠牲にして，大きなビット線ドライバをセル内に配置する方針である．前者は，キャッシュなど，その容量が大きいときに，後者は，レジスタ・ファイルなど，容量が小さいときに有利である．

今回の評価では後者を採用した．実際のレイアウトでは，セル面積の半分以上をビット線ドライバが占めることとなった．文献 1) では前者を採用しているが，この程度のサイズでは後者の方が実際に高速であった．

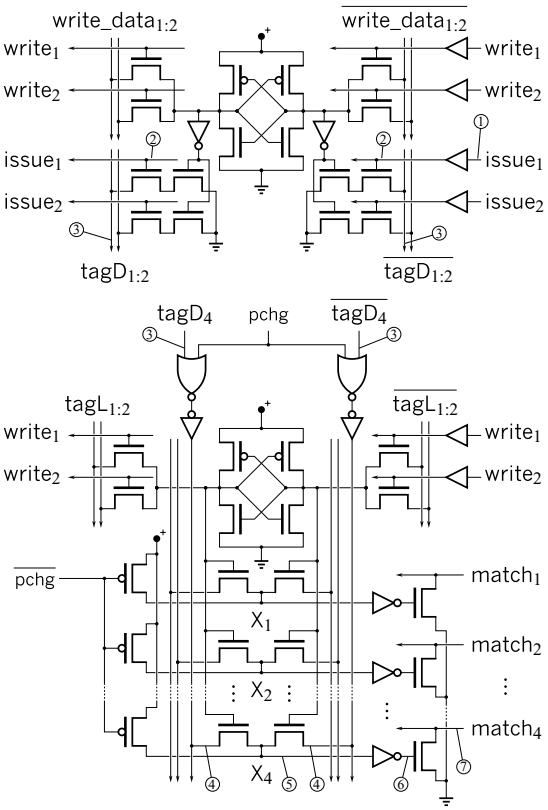

図 10 従来方式の RAM セル（上）と CAM セル（下）
Fig. 10 RAM cell and CAM cell of usual schemes.

5.1.2 CAM

図 10 下に，CAM セルの回路図を示す．セルの上半分は tagL/R を記憶する RAM セルである．この RAM セルは，上述した tagD 用の RAM セルと，レイアウト上の差異はあるものの，基本的には同じものである．

下半分は比較器のアレイである．図では，$IW_q = 2$，$IW = 4$ としている．そのため，書き込みポート数は 2 でよいが，比較器は 4 組必要である．

今回用いた比較器は，2 個 1 組の nMOS パス・ゲートによってダイナミックなセレクタを構成するものである．2 個のパス・ゲートのうち，4T セルの状態によってどちらか一方が ON になっている．4T セルの記憶する値と比較入力の値が異なる場合には，ON となっているパス・ゲートに接続された比較入力線が low となり（④），出力線（⑤）にプリチャージされた電荷が放電される．TW 個のセルのうちのいずれかが不一致を検出すると，そのセルの nMOS マッチ線ドライバが TW 個のセルにわたってひかれたマッチ線を pull-down する（⑥〜⑦）．IW 本のマッチ線の出力を OR することにより，最終的な結果を得る．

なお，比較器の基本的な構成法には，今回用いたパ

ス・ゲートによるセレクタを用いる方法のほかに，複合ゲートによるXORを用いる方法がある．それぞれに対して，スタティック/ダイナミック回路による構成が考えられる．さらにダイナミック回路の場合には，マッチ線ドライバを別途用意せず，セレクタ，あるいは，XORゲートで，マッチ線を直接ドライブする方式がある．合計すると，6通りの構成法が考えられる．上記の方式は，これらのうち，ドライバを別途用意するダイナミックなセレクタによる方式にあたる．また文献1)では，ドライバを別途用意しないダイナミックXORを用いる方式が示されている．今回は，6とおりの方式を予備評価して，最も高速であった上記の方法を選択し，詳細に設計した．ただし，それぞれの遅延の差は5%程度でしかなかった．

5.1.3 DMT

DMTのセルの構成は，すでに図5に示した．DMTセルは，上述した従来方式のtagD用RAMセルと基本的には同じものであるが，以下の点が異なる：

(1) 3.5節で述べたように，ビット線は横方向に，ワード線は斜め方向に引かれる．ワード線長，ビット線長はWS_q，TWとは独立に，DMTの幅によって決まる．

(2) 読み出しポート数は，IW_qにかかわらず1でよい．前述したように，今回用いた構成では，ビット線ドライバがセル面積の半分以上を占める．したがって，読み出しポート数が，セル面積，ひいては，読み出しの遅延に与える影響は無視できない．

5.1.4 *select*

*select*ロジックとしては，Prefix-sum方式を試した．この方式は，自分より優先順位の高い要求の数，すなわち，要求のprefix sumを求めることにより，自らが何番目に選ばれるかを判断する．なお今回は，簡単のため，優先順位は固定としている．文献9)には，命令キューのエントリをサイクリックに使用する方式に向けて，優先順位をやはりサイクリックに変化させる方法が示されている．

これらの方式の遅延は，IW_qとはほぼ独立に，$O(\log_2 WS_q)$で与えられる．実際，32 → 4の遅延は，16 → 4の$\log_2 32/\log_2 16 = 5/4$倍程度となっている．

5.2 評価結果

5.2.1 回路面積

各回路の面積を**表3**に示す．表中，×1の列は，各回路の1つ分の面積を表す．×2の列は，命令キューが2本の場合にシステム全体で必要となる，RAM/CAMの2つ分，および，DMTの4つ分を表す．

表3 CAMとDMTの面積（$\times 10^3 \mu m^2$）
Table 3 CAM and DMT area ($\times 10^3 \mu m^2$).

構成		R10K×1		R10K×2	
		×1	×2	×1	×2
RAM		7.351	14.702	26.273	52.546
CAM		11.843	23.686	71.520	143.039
DMT	2	1.168	4.673	2.875	11.502
	4	2.336	8.945	5.751	23.003
	8	4.673	18.690	11.502	46.006
	16	9.354	37.416	23.003	92.013
	32	—	—	46.006	184.025

提案方式においても，単に物理レジスタ・ファイルにアクセスするために，その番号，すなわち，タグを記憶するため，従来方式のRAMと同じものが必要である．したがって，従来方式におけるCAMと，DMTを比較することが適当であろう．

DMTの縮小の方式としてdelayを採用した場合には，幅WS_q，すなわち，各列で最大のDMTが必要であり，提案方式の面積は従来方式のそれを超える．一方stallを採用した場合には，DMTの幅によって従来方式の1〜数分の1程度となる．

いずれにしても，面積は$0.1\,\mathrm{mm}^2$のオーダであり，チップ面積に占める割合は無視できるほど小さい．

5.2.2 回路遅延

遅延の測定結果を，**図11**に示す．同図中，上から6本のバーはDMT，次の2本はRAM + CAM，最後の2本は*select*の遅延（ps）を表す．図中の①〜⑦は，図10中の点①〜⑦に対応する．測定条件は，電源電圧1.8 V，温度85 °C（typical）である．

DMTの遅延

DMTの遅延は，tagD用RAMの遅延と比較することができる：

ワード線の遅延 （①〜②）は，基本的には，ビット幅の増加関数で表される．

RAMのビット幅はTW，すなわち，6，または，7であるが，その遅延は，DMTの幅8の場合より大きい．それは，RAMでは，差動出力のために各セルごとに2つのビット線ドライバを駆動しているためである．逆に，DMTでワード線が斜めになっていることの影響は，それに比べると小さい．

ビット線の遅延 （②〜③）は，基本的には，ワード数の増加関数で表される．

ただし，DMTの幅が16の場合のビット線の遅延は，RAMのワード数16の場合のそれとほぼ同じ値となっているが，DMTの幅が32の場合には，RAMのワード数32の場合のそれよりかなり大きい．それは，幅32のDMTでは，ビット幅も32あり，ワード線の負

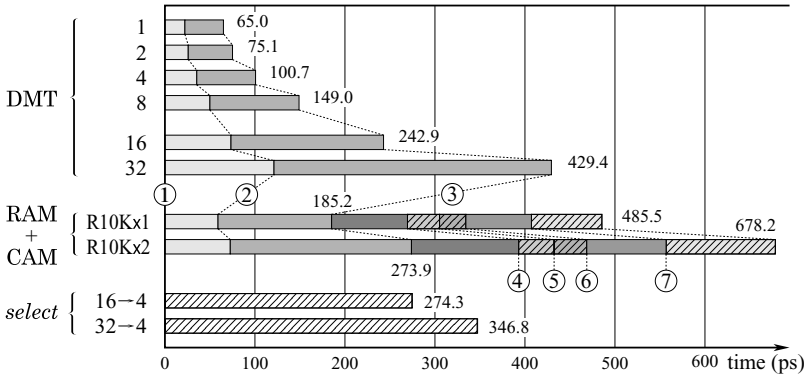

図 11 各回路の遅延
Fig. 11 Delay time of the circuits.

荷が重いためである．そのため，ワード線の電位の上昇が遅く，ビット線ドライバもゆっくりとしか ON にならない．

なお，DMT のパラメータは幅 4〜8 に合わせて最適化されており，幅 16〜32 はそれを単純に拡大したものとなっている．したがって幅 16〜32 に対しては，比較的大きな改善の余地がある．

***wakeup* + *select* の遅延**

DMT の縮小を行わない場合には，構成 R10K×1/R10K×2 では，DMT の幅は 16/32 となる．すなわち，従来方式の 2 本のバーと DMT の一番下の 2 本のバーをそれぞれ比較すればよい．DMT の縮小を行う場合には，従来方式の構成にかかわらず，DMT のその幅のバーと比較すればよい．

構成 R10K×1 の場合，提案方式の *wakeup*+*select* の遅延は，従来方式のそれの 68.0%となる．この場合，提案方式の動作周波数の上限は 1.93 GHz となる．DMT の幅を 4 まで縮小すると，前節の結果から，IPC は 3%程度低下するが，*wakeup* + *select* の遅延は 375.0 ps，動作周波数の上限は 2.7 GHz に達する．

同様に R10K×2 の場合では，提案方式の *wakeup*+*select* の遅延は，従来方式のそれの 75.7%となる．提案方式の動作周波数の上限は 1.29 GHz となる．DMT の幅を 8 まで縮小すると，delay 方式ならば IPC はやはり 3%程度低下するが，*wakeup* + *select* の遅延は 495.8 ps，動作周波数の上限は 2.02 GHz となる．

6. 関 連 研 究

スコアボーディングと Tomasulo アルゴリズム

動的命令スケジューリングの歴史は，**スコアボーディング**[10]と **Tomasulo アルゴリズム**[11]まで遡ることができる．それ以来 *wakeup* は，基本的には，タグによる連想検索によって行われてきた．

ただし，2.1 節で触れたように，タグの選び方にはいくつかのバリエーションがある．タグは個々のデータに付される ID であるから，バックエンドに同時に存在するデータを一意に特定できるものであれば，その役割を果たすことができる．タグという術語を最初に用いた Tomasulo アルゴリズムでは，基本的には，各命令が格納されたリザベーション・ステーションのエントリの番号をタグとしている．Control Data CDC 6600 のスコアボーディング[10]では，出力依存を解消しないため，データを生成する機能ユニットの番号をタグとして用いることができた．投機的実行を行う最近のスーパースケーラでは，投機的な実行結果を格納するバッファのエントリの ID をタグとして用いることが多い．

90 年代後半に入ると，動的命令スケジューリング・ロジックの複雑さがクロック速度に与える影響が懸念されるようになり[1]，複雑を軽減する研究が次第に注目されるようになってきた．たとえば Henry らは，回路上の工夫によって，大規模な命令ウィンドウを検討している[9]．一方，マイクロアーキテクチャ・レベルの改良としては，連想検索の対象を制限したり，連想検索をクリティカル・パスから外すアプローチと，連想検索を完全に省略するアプローチがある．

連想検索の対象を制限する方法

Palacharla らは，**Dependence-Based 命令ウィンドウ**を提案している[3]．この方式では，命令ウィンドウを，演算器クラスタに対応して，複数の小規模な FIFO に分散化する．依存関係にある命令は同じ FIFO に入れられるので，*wakeup* される命令は各 FIFO の先頭に限定することができる．

この方式の潜在的な問題点は各 FIFO への命令の分配（steering）の方法にある．分配はヒューリスティクスに基づいて行われ[3),12),13)]るため，性能はその精

度に依存する．あまり複雑なヒューリスティクスを用いると，クロック速度に対して悪影響を与えかねない．

連想検索をクリティカル・パスから外す方法

Canal らは，**First-Use 法**と，**Distance 法**を提案している[14]．

First-Use 法では，命令ウィンドウは，データの揃っている命令が格納される Ready Queue と，物理レジスタの各エントリを参照する最も古い命令が登録されている First-Use Table から構成される．命令が実行されると First-Use Table が参照され，登録されている命令が Ready Queue へと移される．この方式は，連想検索を不要としているが，従来方式に対する IPC の低下が大きい．性能低下を補うためには，I-Buffer と呼ぶ連想メモリを必要としている．

Distance 法の命令ウィンドウは，データが利用可能になる時刻を管理する Register Availabel Table，実行レイテンシが未定の命令を蓄えている Wait Queue，命令を VLIW 形式にして保持している Issue Queue から構成される．命令は Issue Queue から発行されるので，連想検索を必要としない．しかし，Wait Queue は，連想メモリで構成されている．

Dualflow

連想検索が必要となる本質的な原因は，命令間の依存関係が暗示的に与えられるためである．したがって，連想検索を排除する鍵は，命令間の依存関係を明示的に示すことにある．

Dualflow [15),16)]は，制御駆動とデータ駆動を融合した命令セット・アーキテクチャである．Dualflow は，命令セット・アーキテクチャのレベルから命令間の依存関係を明示的に示すことによって，マイクロアーキテクチャでの連想検索を省略することを目的に考案された．通常の制御駆動アーキテクチャがレジスタを介して間接的にデータの授受を行うのに対して，Dualflow は，レジスタを定義せず，生産側の命令が消費側の命令を明示的に指定することで直接的にデータの授受を行う．消費側の命令は，生産側の命令のフィールドに埋め込まれた数個のポインタによって指定される．

Dualflow の初期の研究では，各命令に埋め込まれた消費側命令へのポインタをほぼそのままの形で保持する，ポインタ形式の命令ウィンドウを仮定していた[15]．ポインタ形式では，従来のスーパースケーラでは CAM で構成されるテーブル rdyL/R を RAM に置き換えることができる．*wakeup* では，まず RAM からポインタを読み出し，それをアドレスとして rdyL/R を格納する RAM にアクセスする．

Dualflow は，しかし，命令セット・アーキテクチャの制限が強すぎるため，無用な命令によってコード・サイズが爆発するという問題点があった．この問題のため，Dualflow それ自体は成功しなかったが，その後の研究に影響を与えることとなった．

EDF 命令ウィンドウ

佐藤らは，本稿で述べたのと同様に，スーパースケーラのフロントエンドにおいて命令間の依存関係を明示的な形式に変換しておく方法を提案している[17]．彼らが用いる **EDF（explicit data forwarding）命令ウィンドウ**は，Dualflow の初期のポインタ形式と基本的に同じものである．

ポインタ形式には，しかし，以下の問題点がある：

(1) メリットが小さい．

rdyL/R を格納するテーブルのアクセス部分が，CAM の比較器から，RAM の行デコーダに置き換わるのだが，その遅延の差はそれほど大きくはない．

(2) 保持できるポインタの数の制限が，IPC とハードウェアの複雑さの間にトレードオフを生じる．

ポインタの数に対する制限は，Dualflow ではコンパイラによって解決されるのに対して，スーパースケーラではハードウェアによって解決されなければならない．そのため現状では，IPC とハードウェアの複雑さの間に厳しいトレードオフが生じている．特に，投機失敗時の状態回復処理が難しくなる．

DMT

ポインタ形式においては，読み出されたポインタで rdyL/R を格納する RAM にアクセスする際に，その RAM の行デコーダにおいてポインタのデコードが行われる．このデコード処理をフロントエンドで済ましておくという着想から，DMT は元々Dualflow 向けの *wakeup* ロジックとして考案された[16]．前述したポインタ形式の 2 つの問題点は，本稿で述べたように，DMT でほぼ完全に解消される．

上述したの既存方式に対する提案方式の定性的な優位点は，以下のようにまとめられる：

(1) 構造がシンプルである．

提案方式では，通常の RAM とほとんど変わらない DMT のみによって *wakeup* を実現することができる．

(2) IPC に対するペナルティはごく小さい．

- DMT の縮小を行わなければ，IPC に対するペナルティはほとんどない．
  DMT の縮小は，遅延とペナルティとのトレードオフを導入するが，その関係は厳しくない．
- 投機失敗時の状態回復処理が容易であり，DMT に関連して新たなペナルティが生じることはない．

7. おわりに

本稿では，命令間の依存関係を直接的に表す行列DMTを用いた動的命令スケジューリング方式について述べた．本方式では，小容量のRAMを読み出すことで *wakeup* を実現することができる．

富士通株式会社から提供された0.18 μm CMOSプロセスのデザイン・ルールに基づいてレイアウト設計を行い，Hspiceを用いて遅延を計測した．その結果，MIPS R10000と同様の構成において，動的命令スケジューリング・ロジックの遅延は516.3 psとなり，従来方式の68.0%にまで削減された．その場合の動作周波数の上限は1.93 GHzとなり，この部分がクリティカルとなる可能性はきわめて低いといえる．

加えて本稿では，DMTを縮小することによってその遅延をさらに短縮する手法についても述べた．この手法では，*wakeup* の遅延をIPCに対するペナルティに転化することができる．シミュレーションによりペナルティを測定したところ，DMTを1/4にまで縮小しても，IPCの悪化は3%以下であることが分かった．R10000の2倍の計算資源を持つ仮想的な8-wayの構成においても，DMTの縮小によって2.02 GHzの動作周波数の上限を達成することができる．

しかし今回用いた0.18 μmのプロセスでは，従来方式であっても，*wakeup* + *select* の遅延がクリティカルになるかどうかは微妙なところである．また提案方式では，DMTの縮小を行うまでもなく，それがクリティカルになる可能性は低いという結果となった．

しかしDMTの縮小には，配線遅延の影響を緩和する効果がある．図11のグラフ中，斜線をかけてある部分はほぼゲート遅延のみからなるが，その他は配線遅延の影響を強く受ける．グラフからは，特に *wakeup* で配線遅延の影響を受ける部分の割合が大きく，その遅延がスケーリングされにくいことが容易に想像される．一方DMTでは，幅16〜32では配線遅延の影響が大きいものの，幅が縮小されるにつれてその影響は次第に小さくなり，幅2，1ではほとんど0になる．したがって，LSIがより微細化され，配線遅延の影響がより大きくなった場合には，DMTの縮小が効果を発揮する可能性が高い．したがって今後は，より微細なプロセスで本方式の有効性を確認していく予定である．

謝辞 富士通株式会社には，LSIの設計情報をご提供いただいたので，ここに深甚なる謝意を表したい．本研究の一部は文部省科学研究費補助金，基盤研究（B）（2）#12480072，#12558027，#13480083による．

参考文献

1) Palacharla, S., Jouppi, N.P. and Smith, J.E.: Quantifying the Complexity of Superscalar Processors, Technical Report, Univ. of Wisconsin-Madison (1996).
2) Keller, J.: The 21264: A Superscalar Alpha Processor with Out-of-Order Execution, *Proc. 9th Annual Microprocessor Forum* (1996).
3) Palacharla, S., Jouppi, N.P. and Smith, J.E.: Complexity-Effective Superscalar Processors, *Proc. 24th Int'l Symp. on Computer Architecture* (*ISCA24*) (1997).
4) Yeager, K.C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, No.4, pp.28–40 (1996).
5) Harris, D. and Horowitz, M.: Skew-tolerant domino circuits, *IEEE J. Solid-State Circuits*, Vol.32, No.11, pp.1702–1711 (1997).
6) SPARC International Inc.: The SPARC Architecture Manual, Version 9 (1994).
7) 五島正裕ほか：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，情報処理学会研究報告98–ARC–130 (SWoPP '98), pp.115–120 (1998).
8) Burger, D., Austin, T.M. and Bennett, S.: Evaluating Future Microprocessors: The SimpleScalar ToolSet, Technical Report CS-TR-1308, Univ. of Wisconsin-Madison (1996).
9) Henry, D.S., Kuszmaul, B.C., Loh, G.H. and Sami, R.: Circuits for Wide-Window Superscalar Processors, *Proc. 27th Int'l Symp. on Computer Architecture* (*ISCA27*) (2000).
10) Thornton, J.E.: Parallel operation in the Control Data 6600, *Proc. AFIPS Fall Joint Computer Conf. Part II*, Vol.26, pp.33–40 (1964).
11) Tomasulo, R.M.: An effective algorithm for exploiting multiple arithmetic units, *IBM J.*, Vol.11 (1967).
12) Farkas, K.I., Chow, P., Jouppi, N.P. and Vranesic, Z.: The Multicluster architecture: Reducing cycle time through partitioning, *Proc. 30th Int'l Symp. on Microarchitecture* (1997).
13) Canal, R., Parcerisa, J.M. and González, A.: Dynamic cluster assignment mechanisms, *Proc. 6th Int'l Symp. on High-Performance Computer Architecture* (*HPCA6*) (2000).
14) Canal, R. and González, A.: A low-complexity issue logic, *Proc. 14th Int'l Conf. on Supercomputing*, pp.327–335 (2000).
15) 五島正裕ほか：Dualflowアーキテクチャの提案，*JSPP2000*, pp.197–204 (2000).
16) 五島正裕ほか：Dualflowアーキテクチャの命令発行機構，情報処理学会論文誌，Vol.42, No.4,

pp.652–662 (2001).
17) 佐藤寿倫，有田五次郎：連想検索を取り除いたスーパースカラプロセッサ向け命令発火機構，*JSPP2001*, pp.23–30 (2001).
18) 西野賢悟，五島正裕ほか：スーパースケーラのための高速な動的命令スケジューリング方式の改良，JSPP2001（ポスター），pp.137–138 (2001).

(平成 13 年 2 月 8 日受付)
(平成 13 年 5 月 24 日採録)

五島　正裕（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学大学院情報学研究科助手．高性能計算機システムの研究に従事．

西野　賢悟（学生会員）

1978 年生．1997 年愛知県立旭丘高校卒業．2001 年京都大学工学部情報学科卒業．同年より同大学大学院情報学研究科修士課程に進学．プロセッサ・アーキテクチャの研究に従事．

グェン　ハイハー

1974 年生．1993 年ハノイ大学工学部退学．1999 年京都大学工学部情報工学科卒業．2001 年同大学大学院情報学研究科修士課程修了．同年（株）京都ソフトウェアリサーチ入社，フラッシュ・ファイル・システムの開発に従事．

縣　　亮慶

1977 年生．1997 年国立舞鶴工業高等専門学校卒業．1999 年京都大学工学部情報学科卒業．2001 年同大学大学院情報学研究科修士課程修了．プロセッサ・アーキテクチャの研究に従事．

中島　康彦（正会員）

1963 年生．1986 年京都大学工学部情報工学科卒業．1988 年同大学大学院修士課程修了．同年富士通（株）入社．工学博士．1999 年京都大学総合情報メディアセンター助手．同年同大学大学院経済学研究科助教授．計算機アーキテクチャに興味を持つ．IEEECS，ACM 各会員．

森　眞一郎（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1989 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．1992 年同大学院総合理工学研究科情報システム学専攻博士課程単位取得退学．同年京都大学工学部助手．1995 年同助教授．1998 年同大学大学院情報学研究科助教授．工学博士．並列/分散処理，計算機アーキテクチャの研究に従事．IEEE，ACM 各会員．

北村　俊明（正会員）

1955 年生．1978 年京都大学工学部情報工学科卒業．1983 年同大学大学院博士課程研究指導認定退学．同年富士通（株）入社．工学博士．2000 年京都大学総合情報メディアセンター助教授．計算機アーキテクチャに興味を持つ．電子情報通信学会，IEEE，ACM 各会員．

富田　眞治（正会員）

1945 年生．1973 年京都大学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．1981 年京都大学工学部情報工学科教授．1998 年同大学大学院情報学研究科教授．並列計算機システムに興味を持つ．著書「並列計算機構成論」,「並列処理マシン」,「コンピュータアーキテクチャI」等．電子情報通信学会，IEEE，ACM 各会員．