

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[†] ゲンハイハー[†]
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、オペランドの有効性を追跡する *wakeup* と呼ぶロジックを持つ。本稿では、従来の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 RAM を用読み出すことで *wakeup* を実現する方式と、その遅延を IPC に対するペナルティに転化する手法を示す。実在する .18 μ m CMOS プロセスのデザイン・ルールに基づいてこ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、2GHz を越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. This paper describes a new scheduling scheme which substitutes association of tags by reading a small RAM which directly represents dependence between instructions, and changes the delay of the logic into IPC penalty. We actually designed the logic guided by a design rule of a real .18 μ m CMOS process, measured the areas, and calculated the delays by Hspice. And we also evaluated the IPC penalty by simulation. The evaluation result shows that this scheme achieves over 2GHz clock speed with the IPC penalty less than 3%.

1. はじめに

現在のスーパースケーラでは、クロック速度が命令発行幅 (*IW*:Issue Width) とウィンドウ・サイズ (*WS*) を制限する主因となりつつある。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、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 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、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* は、LSI の微細化、パイプラインの深化にともなってクリティカルになっていくと予測されるのである。

このような背景から我々は、*wakeup* を高速化する全く新しい動的命令スケジューリング方式を提案した²⁾。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* を実現するため、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、この方式を更に高速化する手法と、それらの定量的評価の結果を示す。

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。

[†] 京都大学 情報学研究所
Graduate School of Informatics, Kyoto University
^{††} 京都大学 経済学研究所
Graduate School of Economics, Kyoto University
^{†††} 京都大学 総合情報メディアセンター
Center for Information & Multimedia Studies, Kyoto University

2. 従来の動的命令スケジューリング方式

スーパースケ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全ての遅延は IW, WS の増加関数で与えられる。ただしそれらの遅延は、パイプラインやクラスタリング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。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、その理由について詳しく述べる。以下まず 2.1 節においてスーパースケ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 従来の動的命令スケジューリング方式

Out-of-order スーパースケラは、論理的なレジスタとは別に、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。このバッファの構成方式には、リオーダー・バッファを用いる方式と、物理レジスタを用いる方式がある。スーパースケラにおける動的命令スケジューリングは、これらのバッファのエントリを用いて、局所的にデータ駆動型の計算を行うこととみなすことができる。命令ウィンドウ内で、命令 I_p が生産するデータを命令 I_c が消費する場合を考えよう。バッファのエントリを介して I_p から I_c にデータが渡されることに着目すると、スケジューリングの処理は以下のように説明できる：

(1) **rename** 命令がフェッチされると、論理レジスタ番号からタグへの変換が行われる。

I_p には、バッファの 1 エントリが割り当てられ、エントリはデータが『ない』状態に初期化される。このエントリの ID がタグである。 I_p に割り当てられたタグを特に tagD ということにする。

I_c は、左/右のソースの論理レジスタ番号から、依存する I_p に割り当てられた tagD を得る。これを tagL/R ということにする。 I_c は、tagL/R で示されるエントリにデータが書き込まれるのを待つ。

(2) **wakeup** I_p の実行にともなって、 I_c が実行可能になることを検出する。

I_p が実行されると、その結果は tagD で示されるエントリに書き込まれ、エントリはデータが『ある』状態に遷移する。

I_c は、tagL/R で示されるエントリにデータが『ある』のを見て、発行可能になる。

(3) **select** 発行可能な命令から、実際に発行するものを選択し、発行する。

2.2 命令スケジューリングのパイプライン化

次に、**rename**, **wakeup**, **select** の各処理をパイプライン化することを考えよう。命令パイプライン中のステージの違いから、**rename** と (**wakeup+select**) と

に分けて考える必要がある。

rename は、必要ならば、パイプライン化することでクリティカル・パスから外することができる。実際現存するスーパースケラでは、**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。ただしもちろん、その分だけ分岐予測ミス・ペナルティが増加することになる。

wakeup と **select** は、**rename** とは異なり、パイプライン化することができない。**wakeup** と **select** のそれぞれに 1 サイクルかけた場合、先行する命令の結果を消費する命令は、先行する命令に引き続くサイクルに実行することができない。このことは、レイテンシが 1 サイクルである演算器——通常構成では ALU——からはオペランド・パ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、クロック速度の向上に見合わない可能性が高い。このような観点から、**wakeup** と **select** は、実際上、合わせて 1 サイクルで実行しなければならないとできる。

2.3 Superscalar の wakeup

wakeup は、実行される I_p の tagD を RAM から読み出し、それをキーとして CAM にアクセスするという処理によって実現される。

前述したように、**wakeup** と **select** と合わせて 1 サイクルで実行する必要がある。これらのうち、**select** の遅延は専らゲート遅延からなるため、LSI の微細化に伴って順調に短縮されていくと予測される。一方、**wakeup** の遅延は RAM と CAM のワード線、ビット線などの配線遅延からなるため、LSI の微細化の恩恵を受けにくい。以上の理由により **wakeup** は、LSI の微細化、パイプラインの深化にともなっていくそうクリティカルになっていくと予測されるのである³⁾。

命令ウィンドウは、実際には、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、あるいは、命令キューによって実現されることが多い。例えば MIPS R10000 では、ロード/ストア、整数演算、浮動小数点演算のそれぞれに 16 エントリの命令キューを用意している³⁾。しかし複数のキューによって実現される場合にも、それらの基本的な構成は変わらず、命令キューの命令発行幅を $IW_q (< IW)$ 、サイズを $WS_q (< WS)$ とすると、各キューにおける RAM、CAM のワード数は WS から WS_q に、また、RAM の読み出しポート数は IW から IW_q に、それぞれ縮小することができる。しかし CAM の比較入力ポート数は、 IW のまま縮小することができない。

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、従来のタグによる連想処理に基づくものとは根本的に異なり、各命令間のデータ依存関係を直接的に表すデータ構造を用い

てスケジューリングを行う。本章では、提案方式について詳しく述べる。

3.1 DMT

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 matrix 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。

3.1.1 DMT の概要

図 1 に、DMT の概念図を示す。DMT は、基本的には、rdyL/R 用に各 1 つずつの WS 行 WS 列の RAM である (ただし、対角要素は使用しない)。それぞれの RAM の各行は I_p に、各列は I_c に対応する。各要素は、対応する I_p と I_c の間の依存関係を表す。すなわち、 p 行 c 列の要素は、命令ウィンドウ内のエントリ $ID=p$ の I_p の実行結果を $ID=c$ の I_c が消費するなら“1”、そうでなければ“0”とする。図 1 の例は、連続する 4 つの命令が $ID=1, 2, 3, 4$ のエントリに順に格納された場合を表している。図の例では、 $ID=1$ の命令が生産する $r1$ を $ID=2, 3$ の命令がそれぞれの左オペランドとして消費している。したがって DMT では、1 行目の 2, 3 列目が“1”となる。その他の要素も同様に求められる。

DMT の更新は、従来のスーパースケラの *rename* ステージにおいて実行しておくことができる。本稿では提案手法のポイントとなる *wakeup* の処理について述べる。

wakeup ステージにおいては、実行される最大 IW 個の I_p に対応する IW 行の OR を求めれば、セットすべき rdyL/R を表すベクトルを求めることができる。例えば、図 1 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。この命令の実行結果は、 $ID=2, 3$ の命令が左オペランドとして消費する。自明ではあるが、DMT の第 1 行は $ID=1$ の命令が実行される時にセットすべき rdyL/R を表している。実際に $ID=1$ の命令が実行されると、 $ID=2, 3$ の rdyL がセットされ、次のサイクルにはその 2 つの命令が実行可能になる。実際にその $ID=2, 3$ の命令が選択され同時に実行される場合には、第 2 行と第 3 行の OR を求めればよい。結果、 $ID=4$ の命令の rdyL/R がそれぞれによってセットされる。ただし実際には、OR

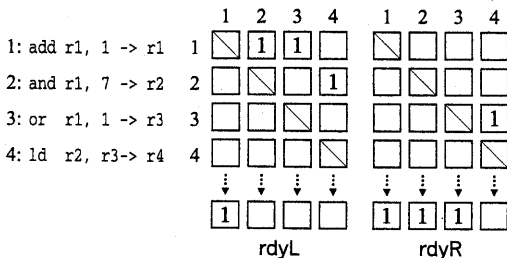


図 1 依存行列テーブル (dependence matrix table:DMT)

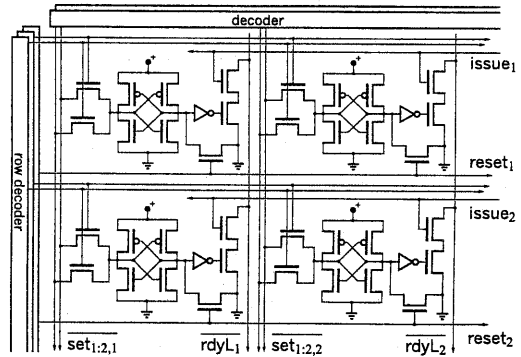


図 2 依存行列テーブルのロジック (rdyL 用)

を求めると言っても、各列で“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 I_p はただ 1 つだからである。

命令ウィンドウが q 本の命令キューによって実装される場合には、生産側と消費側に対応して、 $q \times q$ 個の DMT を用意すればよい。それぞれの DMT は、 $WS \times WS$ word から、 $WS_q \times WS_q$ word に縮小される。

3.1.2 DMT の実装

DMT を実装するにあたっては、単に 1-read の RAM を用いればよい。複数行の OR を求めるための特別なロジックは必要ない。

図 2 に、DMT の回路図を示す。同図には、左上の 2 行 2 列分のセルが示してある。

各セルの中央にある 4T セルの左側は、書き込みポートである。DMT の更新は、上書きではなく、ビット・セットを行う必要がある。そのため書き込みポートは、single-bitline となっている。bitline が low であればセットされるが、high であってもリセットされない。

その一方で DMT のエントリは、使用に先だってエントリごとリセットしておく必要が生じる。そのため、4T セルの右下にリセット用のポートが用意されている。このリセットのため、エントリの解放後再利用するためには、実装によっては 1 サイクルの空きが生じることがある。

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。図からも明らかのように、この読み出しポート部は、single-bitline の 1-read の RAM と構造上は全く変わらない。

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点が異なる。各ビット線 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 OR を読み出すことができる。

3.2 DMTの縮小

依存する I_p と I_c の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⁴⁾。本節では、この性質を利用して、前節で述べた *wakeup* を更に高速化する手法について述べる。

前節で述べた DMT は $(WS - 1) b \times WS$ word であったが、このビット数を w ($0 \leq w < WS - 1$) b に削減することによって *wakeup* の遅延を更に短縮することを考える。この w を DMT の幅と呼ぶことにする。依存する命令間の距離が DMT の幅以下の場合には、DMT によって $rdyL/R$ を更新する。幅を越えていた場合には、クロック速度に影響を及ぼさない別の方法を用い、IPC に対するペナルティに転化するのである。依存する命令間の距離は短い場合が多いので、ペナルティの影響は小さいと予想される。

命令間の距離が DMT の幅を越える場合への対応としては、以下の 2 つの方法が考えられる：

dealy 幅 w ($w < WS - 1$) と $WS - 1$ の 2 種の DMT を用意し、後者には 1 サイクル余分にかける。

距離が幅 w を越えている場合には $rdyL/R$ の更新が 1 サイクル遅れるため、 I_c は I_p に引き続き 1 サイクルには発行されない。

stall 依存する I_p までの距離が DMT の幅を越えている場合には、 I_c のデコードの段階でフロント・エンドをストールさせる。

I_p の実行が終了した後にデコードを再開すれば、 $rdyL/R$ は 1 に初期化される。

DMT の縮小の方法

図 3 に、DMT の縮小の様子を示す。左は元々の、右が縮小後の DMT である。同図では、 $WS = 8$ 、 $w = 4$ である。元々の 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) 集める方法には任意性があるが、よりクリティカルであるビット線の長さを短縮することを優先して、図 3 右のようにするとよいであろう。

図 3 右から明らかなように、縮小された DMT のワード線、ビット線は、それぞれ w 個のセルにしか接続されておらず、それぞれの長さは WS とは無関係となっている。したがって *wakeup* の遅延は、 WS とは

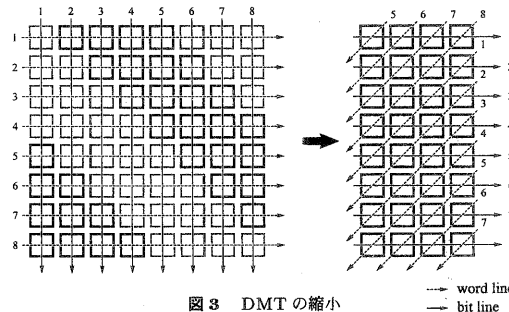


図 3 DMT の縮小

独立に、 w によって決まる。

ウィンドウ・エントリの使用に対する制限

上では、命令間の距離という言葉曖昧に用いていたが、DMT によって $rdyL/R$ が更新できるのは、正確には、命令間の距離ではなく、ウィンドウ内のエントリ間の距離が DMT の幅以下の場合である。したがって、命令流上での距離とウィンドウ内での距離が (ある程度) 一致するように制御する必要がある。

このことは特に、命令ウィンドウを複数の命令キューによって構成する場合に問題となる。エントリをサイクリックに使用する場合、異なる命令キュー間では、パディングによってエントリの消費の歩調を揃える必要がある。例えば R10000 の命令キューの構成では、各キューのサイズは同じであるから、例えばあるサイクルに 2 つの整数命令が整数命令キューに格納されるとすると、他のキューでも 1 つのエントリを無駄に消費する必要がある。

4. 評価

従来方式と提案方式との定量的な比較を行う。比較項目は、3.2 節で述べた DMT の縮小に対するペナルティと、回路の面積/遅延である。前者は 4.1 節で、後者は 4.2 節で、それぞれ述べる。

4.1 IPC の評価

測定条件

SimpleScalar ツールセット (ver.2.0) に対して、3.2 節で述べた DMT の縮小を実装し、SPEC ベンチマークを用いて 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 R10000³⁾ のマシン構成を用いた。R10000 は、ロード/ストア、整数演算、浮動小数点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。1 次キャッシュは、命令/データ、それぞれ、容量 32KB、ライン・サイズ 32B である。2 次キャッシュは命令/データ共有で、容量 1MB、ライン・サイズ 64B 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 2b 飽和型カウンタによるもの (bimod) を用いた。これを、構成 R10K \times 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、 $(IW_q, IW, WS_q, TW) = (4, 8, 32, 7)$ としたものも合わせて測定した。これを構成 R10K \times 2 と呼ぶ。

測定結果

まず、DMT の縮小の前に、前章で述べた DMT エントリのリセットとパディングの影響を調べたところ、それらによる IPC の低下率は平均で、59%、最悪でも

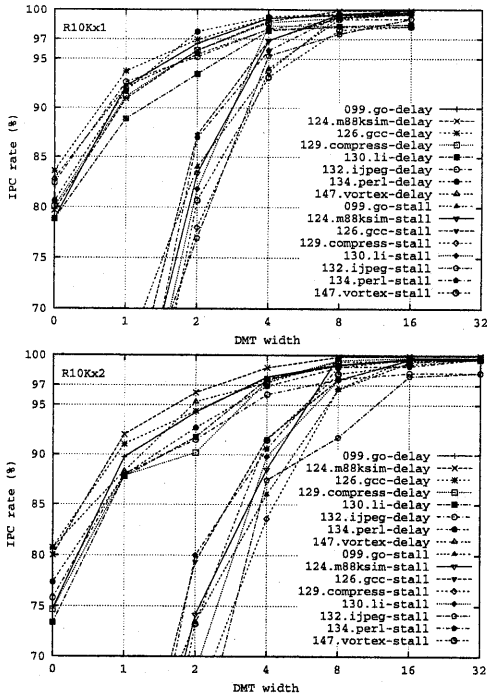


図4 DMTの幅に対するIPC

1.54%であった。後述するDMTの縮小によるIPCの低下率は数%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。

次に、DMTの縮小を行った場合の、従来方式に対するIPCの低下率を図4に示す。なお提案方式には、リセットを組み込んである。また、DMTの幅が WS_q の場合でもパディングを行っている。

結果からは、DMTの幅は、delayならば WS_q の1/4、stallでも1/2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、DMTの幅を0とした場合のdelayの値は、ちょうど、2.2節で述べた、*wakeup*と*select*にそれぞれ1サイクルかけ、ALUからはバイパスを行わない場合に相当する。その場合のIPCは低下は、30%程度になる。

4.2 回路の評価

富士通株式会社から提供された、 $.18\mu\text{m}$ CMOS プロセスのデザイン・ルールに基づいて、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、回路面積を求めた。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。

図5に、従来方式のtagD用RAM、CAMのセルを示す。DMTのセルの構成は、既に図2に示した。

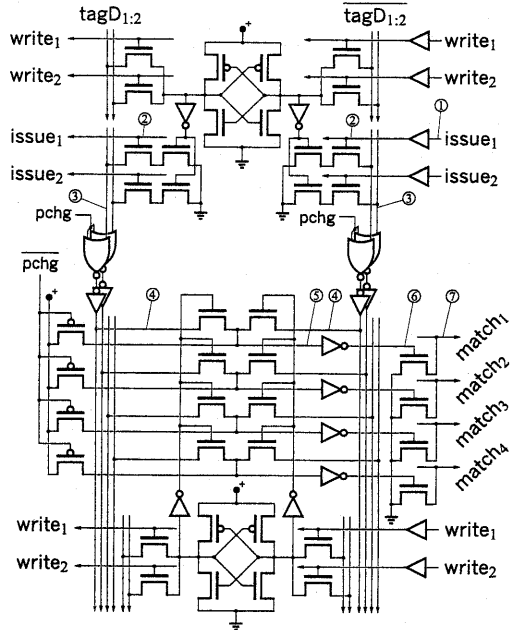


図5 従来方式のRAMセル(上)とCAMセル(下)

面積

各回路の面積を表1に示す。表中、 $\times 1$ の列は、各回路の1つ分の面積を表す。 $\times 2$ 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。

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越える。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 0.1mm^2 のオーダーであり、チップ面積に占める割合は無視できるほど小さい。

表1 CAMとDMTの面積($\times 10^3 \mu\text{m}^2$)

構成	R10K $\times 1$		R10K $\times 2$	
	$\times 1$	$\times 2$	$\times 1$	$\times 2$
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
	4	2.336	8.945	5.751
	8	4.673	18.690	11.502
	16	9.354	37.416	23.003
	32	—	—	46.006
			46.006	184.025

遅延

遅延の測定結果を、図6に示す。測定条件は、電源電圧1.8V、温度85°C (typical)である。同図中の①～⑦は、図5中の点①～⑦に対応する。⑦より右は、IW本のマッチ線のORをとる部分の遅延である。

また、16→1、16→2、および、16→4、32→4のselectの遅延も同じ方法で測定した；ただし、 $WS_q \rightarrow IW_q$ は、 WS_q 個の命令から IW_q 個の命令を選択することを意味する。

これらのうち、前2つ ($IW_q = 1, 2$)には、通常利用される、1命令を選択するアービタをカスケード接続した方式¹⁾を用いた。グラフにも示されているように、この方式の遅延は IW_q に比例するため、 $IW_q \geq 3$ での利用は難しい。

そこで後2つ ($IW_q = 4$)には、発行要求のprefix-sumを計算する方式を用いた。紙面の都合上詳しくは述べられないが、prefix-sum方式では、全体をただか $\log_2 WS_q + 1 (= 5 \sim 6)$ 段のドミノ・ゲートで構成することができるため、カスケード方式を越える高速性を達成することができた。prefix-sum方式の16→4の遅延は、カスケード方式の16→1のそれに比肩する。

MIPS R10000は、wakeupはR10K×1、selectは16→2にあたる。その場合、wakeupとselectの遅延の合計は986.4psである。この部分が決める動作周波数の上限は1.01GHzとなり、現存する.18μmの4-way スーパースケーラの仕様とおおよそ符合する。

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、従来方式の構成に関わらず、DMTのその幅のバーと比較すればよい。

構成R10K×1の場合、wakeup+selectの遅延は従来方式の68.0%、動作周波数の上限は1.93GHzとなる。DMTの幅を4まで縮小すると、前節の結果からdelay方式ならばIPCは3%程度低下するが、遅延は375.0ps、動作周波数の上限は2.7GHzに達する。

同様にR10K×2の場合では、wakeup+selectの遅延は、従来方式の75.7%、動作周波数の上限は

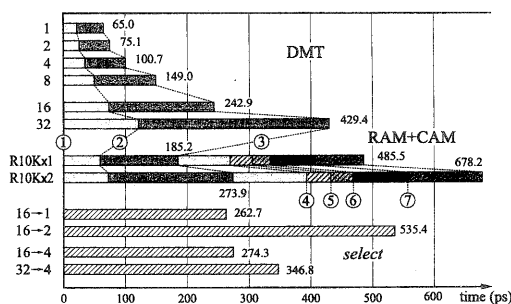


図6 各回路の遅延

1.29GHzとなる。DMTの幅を8まで縮小するとIPCはやはり3%程度低下するが、遅延は495.8ps、動作周波数の上限は2.02GHzとなる。

5. おわりに

本稿では、命令間の依存関係を直接的に表すテーブル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を読み出す程度の処理で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.18μm 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、ロジックの遅延は516.3ps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1.93GHzとなり、動的命令スケジューリングがクリティカルとなる可能性は極めて低いと言える。

加えて本稿では、DMTを縮小することによってその遅延をIPCのペナルティに転化する方法についても述べた。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をウィンドウ・サイズの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。DMTの縮小によって、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、2.02GHzの動作周波数の上限を達成することができる。

以上から、スーパースケーラにおいて、その動的命令スケジューリング・ロジックがクロック速度を規制する可能性は極めて低くなくなったと結論づけることができよう。

謝辞

富士通株式会社には、LSIの設計情報をご提供頂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、同#12558027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) 縣亮慶, ゲンハイハー, 五島正裕, 中島康彦, 森真一郎, 富田真治: Superscalarにおける低遅延な命令発行機構, 情処研報 2000-ARC-139, pp. 109-114 (2000).
- 3) Yeager, K. C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, No. 4, pp. 28-40 (1996).
- 4) 五島正裕, ゲンハイハー, 森真一郎, 富田真治: 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, 情処研報 98-ARC-130, pp. 115-120 (1998).