社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ARC－156　(5)
2004／2／2

データ依存を考慮したプレスケジューリングを行う命令スケジューラ

打田高章[†]　本間幹英[†,☆]　嶋田創[†]
安藤秀樹[†]　島田俊夫[†]

命令ウィンドウを大きくすることで，より多くの命令レベルの並列性（ILP:instruction-level parallelism）を引き出すことができる．しかし，大きな命令ウィンドウは，命令発行論理の遅延を増加させるという問題がある．そこで，本論文では，2 段に分割した命令ウィンドウでスケジューリングを行う命令スケジューラを提案する．提案する命令スケジューラでは，1 段目の命令ウィンドウでデータ依存を考慮したプレスケジューリングを行い，2 段目の命令ウィンドウから命令を発行する．これにより，命令発行論理の遅延時間を短く保ったまま，全体の命令ウィンドウのサイズを大きくすることができる．評価の結果，例えば，32 エントリの命令ウィンドウを 2 段に分割した場合の IPC の低下率は，SPECfp95 平均で 6.7%であった．しかし，命令発行論理の遅延の削減によるクロック周波数の向上も考慮すると，SPECfp95 平均では 5.8%命令スループットを向上させることができる．

An Instruction Scheduler with Dependence-based Prescheduling

Takaaki Uchida,[†] Mikihide Honma,[†,☆] Hajime shimada,[†]
Hideki Ando[†] and Toshio Shimada[†]

A large instruction window can exploit more instruction-level parallelism (ILP). However, it increases delay of the issue logic. In this paper, we propose an instruction scheduler whose instruction window is divided into two portions. The first window preschedules instructions based on data dependence, and the second window issues instructions into the functional units. Our scheduler can increase the instruction window without delay increase. Our evaluation results show that, for example, the IPC with our scheduler of total 32-entry divided instruction window is 6.7% lower than that with the conventional scheduler of the same size of the instruction window on the average of SPECfp95. However, if we consider the clock frequency improvement due to delay reduction, the instruction throughput increases by 5.8% over the conventional single instruction window case.

1. はじめに

スーパスカラ・プロセッサは，命令レベルの並列性（ILP:instruction-level parallelism）を引き出すことで性能を改善してきた．より多くの ILP を引き出す方法の一つに，命令ウィンドウを大きくすることがある．命令ウィンドウを大きくすれば，プログラムのより広い範囲から並列に実行可能な命令を見つけることができ，より多くの ILP を引き出すことができる．しかし，命令ウィンドウを大きくすると，命令発行論理の遅延が増加するという問題がある．現在，命令発行論理の遅延はクリティカルパスの 1 つとなっており，クロック周波数の向上を制限する要因となっている．

一般に，遅延時間の影響を緩和する方法として，パイプライン化がある．命令発行論理のパイプライン化の場合，wakeup 論理と select 論理をパイプライン化することが考えられる．しかし，依存関係にある命令を連続したサイクルに発行するためには，wakeup/select を 1 サイクルで行わなければならない．依存関係にある命令を連続して発行できなければ，IPC（instruction per cycle）が大きく低下してしまう．そのため，それらを単純にパイプライン化することができない．

本論文では，2 段に分割した命令ウィンドウでスケジューリングを行う命令スケジューラを提案する．この命令スケジューラでは，デコードされた命令は，まず 1 段目の命令ウィンドウに書き込まれる．1 段目の命令ウィンドウで移動条件を満たした命令は，2 段目の命令ウィンドウへ移動する．そして，2 段目の命令ウィンドウから命令を発行する．1 段目での処理を**プレスケジューリング**と呼ぶ．このようにして命令ウィンドウを 2 段に分割し，各々で独立した wakeup/select を行うことで，命令ウィンドウの遅延を，総エントリ数が 2 分の 1 である命令ウィンドウと同じにすることができる．しかし，命令を発行することができるのは 2 段目の命令ウィンドウだけであるため，総エントリ数が同じである従来の命令ウィンドウに対して IPC が低下することが予想される．この IPC の低下を抑えるため，プレスケジューリングの際にはデータ依存を考慮し，実行可能となるまでの時間が短い命令ほど早く 2 段目の命令ウィンドウに移動させるようにする．こうすることで，実行可能となるまでの時間が短い命令だけが 2 段目の命令ウィンドウに存在するようになり，効率よく命令を発行することができる．これにより，命令ウィンドウの分割による IPC の低下を抑えることができる．以上

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
☆ 現在，中部日本電気ソフトウェア株式会社
Presently with NEC Software Chubu, Ltd.

により，提案する命令スケジューラは，命令発行論理の遅延時間を短く保ったまま，命令ウィンドウを大きくすることができる．

2 章では関連研究について述べる．3 章では提案する 2 段分割スケジューラについて説明する．4 章では命令発行論理とその遅延時間について説明する．5 章で評価結果を示し，最後に 6 章でまとめる．

2. 関 連 研 究

命令発行論理の遅延の問題を解決する研究がいくつか行われている．これらの研究は主に，命令発行論理の単純にするもの，命令発行論理をパイプライン化するもの，命令ウィンドウを小さくするものに分けることができる．

命令発行論理を単純化する研究として以下のものがある．Palacharla ら[10] は，命令ウィンドウを複数の FIFO で構成し，依存関係にある命令は同一の FIFO に挿入する依存ベースの命令ウィンドウを提案した．Henry ら[7] は，命令発行論理の遅延が $\log(n)$（n = 命令ウィンドウサイズ）のオーダで増加する CSP（cyclic segmented prefix）回路を提案した．五島ら[6] は，命令ウィンドウ内の命令間の依存関係を行列の形で RAM に記憶しておき，この RAM を利用して wakeup を行う方法を提案した．これらの研究はいずれも，命令発行論理を単純な回路に置き換えることで遅延を減少させるものである．

命令発行論理の wakeup/select をパイプライン化する研究として以下のものがある．Stark ら[12] は，2 つ前の依存命令による投機的な wakeup を行うことを提案した．Brown ら[2] は，ready となった命令が select される前に，その命令に依存している命令の wakeup を投機的に行うことを提案した．いずれの方法も，投機ミスが発生する可能性があり，回復動作を必要とする．

命令ウィンドウを小さくする研究として以下のものがある．Canal ら[4] は，一部の命令だけを命令ウィンドウに保持し，他の命令は別のバッファに保持する方法（First-use sheme, Distance Scheme）を提案した．また，プレスケジューリングを用いて命令ウィンドウを小さくする研究として以下のものがある．Michaud ら[8] は，命令ウィンドウに書き込む前に，あらかじめ命令を計算された発行時刻で並べかえておくことを提案した．Raasch ら[11] は，大きな命令ウィンドウを小さな複数の命令ウィンドウに分割し，依存情報と実行レイテンシをもとに，計算された発行時刻を用いて命令をスケジューリングする方法を提案した．これらの方法はいずれも，計算された発行時刻が正しく無かった場合（資源競合やデータキャッシュミスが起こった場合）をうまく対応できない問題がある．

3. データ依存を考慮したプレスケジューリングを行う命令スケジューラ

本命令スケジューラは，1 段目の命令ウィンドウでデータ依存を考慮したプレスケジューリングを行い，2 段目の命令ウィンドウから命令を発行する．

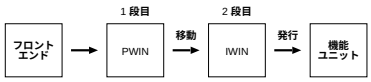
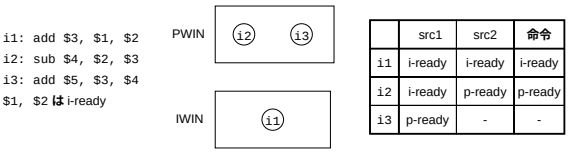


図 1 2 段命令スケジューラ



	src1	src2	命令
i1	i-ready	i-ready	i-ready
i2	i-ready	p-ready	p-ready
i3	p-ready	-	-

図 2 p-ready，i-ready の例

図 1 にスケジューラの概略図を示す．本論文では，命令ウィンドウ間の命令の移動を，単に移動と呼ぶ．1 段目の命令ウィンドウは，プレスケジューリングを行うためのもので，**PWIN（presheduling window）**と呼ぶ．また，2 段目の命令ウィンドウは，命令の発行を制御するためのもので，**IWIN（issue window）**と呼ぶ．フロントエンドでの処理を終えた命令は，PWIN に書き込まれる．PWIN の中でデータ依存に基づくある条件を満たした命令は，IWIN へ移動する．理想的には，実行可能となるまでの時間が短い命令だけが IWIN に存在するようになり，効率よく命令を発行することができる．そのため，命令ウィンドウを分割しつつも，IPC の低下を抑えることができる．

本論文では，命令ウィンドウの構成を，(各命令ウィンドウのエントリ数)x(命令ウィンドウの段数) で表す．例えば，16 エントリの従来の命令ウィンドウは 16x1 となり，PWIN と IWIN が各 16 エントリである命令ウィンドウは 16x2 となる．

3.1 p-ready と i-ready

以降の説明において，命令ウィンドウ内の命令と，そのソース・オペランド（以下 src）の状態を表す言葉として **p-ready**，**i-ready** を用いる．src が p-ready であるとは，src を生成する命令が，PWIN 中に存在しないことである．src が i-ready であるとは，src が利用可能であることである．i-ready な src は p-ready でもある．また，命令が p-ready であるとは，全ての src が p-ready であることである．命令が i-ready であるとは，全ての src が i-ready であることである．i-ready な命令は p-ready でもある．また，IWIN において i-ready である命令は，発行可能である，

図 2 に p-ready，i-ready の例を示す．`i1` は，第 1 ソース・オペランド（src1）と第 2 ソース・オペランド（src2）がともに i-ready であり，命令自体も i-ready である．`i2` は，src1 が i-ready であるが，src2 は依存先の `i1` が既に IWIN に移動しているので p-ready であるため，命令自体は p-ready である．`i3` は，src1 は p-ready であるが，src2 は p-ready でも i-ready でもないため，命令自体は p-ready でも i-ready でもない．

3.2 スケジューリングの方法

プレスケジューリングでは，i-ready となるまでの時間が長い命令は PWIN にとどめ，i-ready となるまでの時間が短い命令ほど早く IWIN に移動させることが望ましい．そこで，p-ready な命令は，そうでない命令よりも i-ready とな

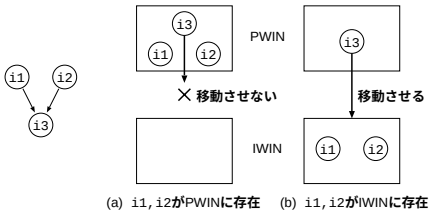


図 3 プレスケジューリングの例

るまでの時間が短いと考え，p-ready でない命令は PWIN にとどめ，p-ready な命令は IWIN に移動させる．

図 3 にプレスケジューリングの例を示す．i3 は，i1 と i2 に依存している．(a) では，i1 と i2 がまだ PWIN に存在しているので，i3 は p-ready ではない．よって，i3 を IWIN に移動させない．(b) では，i1 と i2 が既に IWIN へ移動しているので，i3 は p-ready である．よって，i3 を IWIN へ移動させる．

1 サイクルあたりに PWIN から IWIN へ移動可能な命令数は資源によって制限される．ここでの資源制約は，IWIN の空きエントリ数と移動幅（1 サイクルあたりに IWIN へ移動可能な命令数の最大値）である．IWIN へ移動する命令は，資源制約を満たすように，あらかじめ決められた優先順位にしたがって選ばれる．これは，資源制約が異なる以外は従来の select 論理と同じである．命令が IWIN へ移動することによって，その命令に依存しているオペランドが p-ready となる．これを従来の wakeup と区別するために p-wakeup と呼ぶ．

命令が IWIN に移動した後は，従来の命令ウィンドウと同様にスケジューリングされる．1 サイクルあたりに機能ユニットに発行可能な命令数は資源によって制限される．機能ユニットに発行される命令は，資源制約を満たすように，あらかじめ決められた優先順位にしたがって選ばれる．これは，従来の select 論理と同じである．命令の実行が完了することによって，その命令に依存しているオペランドが i-ready となる．これを従来の wakeup と区別するために i-wakeup と呼ぶ．

3.3 機　　構

2 段命令スケジューラの機構について説明する．**図 4** に概略図を示す．

PWIN と IWIN には従来の命令ウィンドウの機構を利用する．ただし，ready ビットを，i-ready ビットと呼ぶことにする．また，PWIN には，各 src が p-ready であるかどうかという情報を保持する p-ready ビットを付け加える．なお，p-ready をセットするための CAM は，命令ウィンドウを構成する CAM とは別に用意する．

レジスタ・マップ表も従来の機構を利用する．また，それとは別に対応する論理レジスタが p/i-ready であるかどうかという情報を保持する p/i-ready ビットを付け加える．

スケジューリングの手順を説明する．

(1)　PWIN に入る前

命令の src に対応するレジスタ・マップ表のエントリを参照し，p-ready ビットと i-ready ビットの値を得る．命令のデ

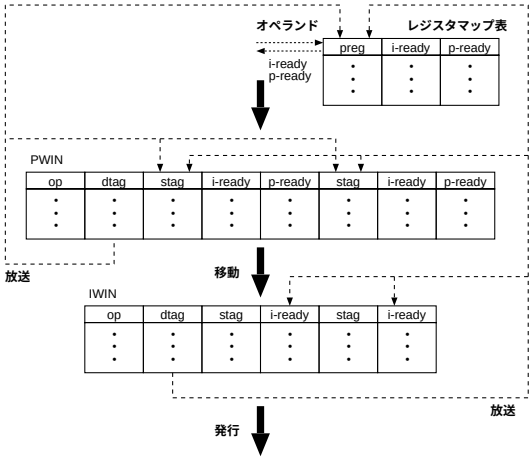


図 4 多段命令スケジューラの機構

スティネーション・オペランド（以下 dst）に対応するレジスタ・マップ表のエントリの p-ready ビットと i-ready ビットに 0 を書き込む．

(2)　PWIN に入るとき

PWIN の空きエントリに従来の命令ウィンドウと同様に命令を書き込む．p-ready ビットと i-ready ビットは，(1) で得た値を書き込む．PWIN に空きエントリがなければストールする．

(3)　PWIN での処理

p-wakeup のために放送されるタグ（(4) 参照）と，各 stag を比較し，タグが一致した src の p-ready ビットを 1 とする．同様に，i-wakeup のために放送されるタグ（(7) 参照）と，各 stag を比較し，タグが一致した src の i-ready ビットを 1 とする．全ての src の p-ready ビットが 1 となった命令の中から，select 論理によって命令を選び，IWIN へ移動させる．

(4)　PWIN から IWIN への移動

移動する命令に依存している命令の p-wakeup を行うため，移動する命令の dst のタグ（以下 dtag）を PWIN に放送する．また，移動する命令の dtag でレジスタ・マップ表を参照し，そのエントリの p-ready ビットを 1 とする．

(5)　IWIN に入るとき

IWIN の空きエントリに従来の命令ウィンドウと同様に命令を書き込む．i-ready ビットには，PWIN での i-ready ビットの値を書き込む．

(6)　IWIN での処理

i-wakeup のために放送されるタグ（(7) 参照）と各 stag を比較し，タグが一致した src の i-ready ビットを 1 とする．全ての src の i-ready ビットが 1 となった命令の中から select 論理によって命令を選び，機能ユニットに発行する．

(7)　機能ユニットに発行されるとき

命令が発行される命令に依存する命令の i-wakeup を行うため，発行される命令の dtag を放送する．また，命令の dtag でレジスタ・マップ表の物理レジスタ番号（preg）を参照し，そのエントリの i-ready ビットを 1 とする．

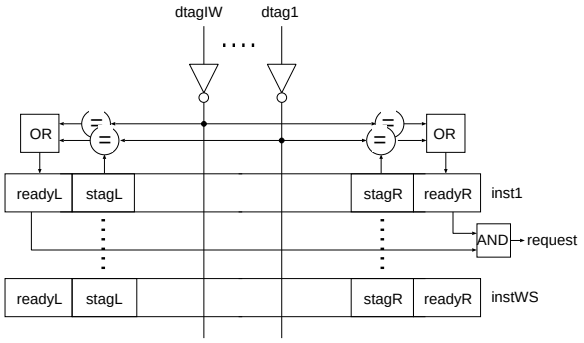


図 5 wakeup 論理

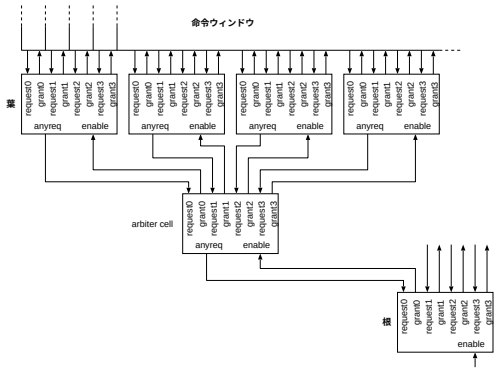


図 6 select 論理

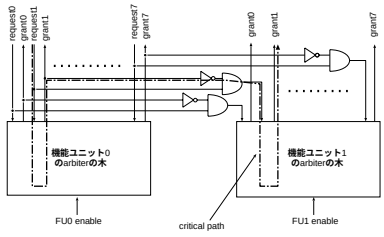


図 7 複数命令発行の場合の select 論理

4. 命令発行論理の遅延時間の見積り方法

命令発行論理は wakeup 論理と select 論理からなる．本研究で用いた命令発行論理は，wakeup 論理に CAM を用いたものを想定している．select 論理は，文献 9) を参考にしている．それぞれの論理の説明と，その遅延時間の見積り方法について以下で述べる．なお，以降の説明で，WS は命令ウィンドウ・サイズを表し，IW は発行幅を表す．

4.1 Wakeup 論理

wakeup 論理は，命令の src の状態を更新し，全ての src が利用可能となった命令の発行の要求を select 論理に送出する．

図 5 に，wakeup 論理を示す．命令が発行されたら，その命令のデスティネーション・タグ (dtag) が命令ウィンドウ内の全エントリに放送される．放送される dtag は最大で IW と同数である．各エントリでは，そのエントリに保持されている命令の 2 つのソース・タグ (stag) のそれぞれと，放送されてきた最大 IW 個の dtag の比較が行われる．比較の結果，各 src につき 1 つでも一致するものがあれば，その src が利用可能であることを示す ready ビットがセットされる．2 つの src の ready ビットがともにセットされたら，select 論理に発行の要求（request）が送出される．

wakeup 論理の遅延は，dtag を駆動する時間 $T_{tagdrive}$，タグの比較を行う時間 $T_{tagmatch}$，比較結果の信号の論理和を取る時間 $T_{matchOR}$ よりなる．$T_{tagdrive}$ は，dtag を駆動する配線の長さと比較器の数で決まる．$T_{tagmatch}$ は，主に比較結果の信号を駆動する配線の長さによって決まる．いずれの配線の長さも CAM セルのサイズに比例し，CAM セルのサイズは WS と IW によって決まる．$T_{matchOR}$ は，論理和への入力数（=IW）によって決まる．よって，wakeup 論理の遅延は WS と IW の関数になる．

なお，dtag を駆動するトランジスタとタグの比較を行うトランジスタの最適なサイズは，WS と IW によって大きく変化する．そのため，各 WS，IW について，トランジスタのサイズを変化させて遅延時間を測定し，最適なトランジスタのサイズを決めた．

4.2 Select 論理

select 論理は，資源制約を満たすように，発行の要求の中から許可するものを選ぶ．発行の要求を許可された命令は機能ユニットに発行される．

図 6 に select 論理を示す．この図では，命令ウィンドウの左の方にある命令ほど優先順位が高い．select 論理は，いくつかの arbiter cell を木の形に連ねることにより実現されている．まず，葉の arbiter cell に，命令ウィンドウの各エントリから送出される request 信号が入力される．request 信号が 1 つでも入力された arbiter cell からは，anyreq 信号が出力される．次に，anyreq 信号が，親の arbiter cell へ request 信号として入力される．同様にして，request 信号は根まで到達する．そして，request 信号が根の arbiter cell まで到達し，資源が利用可能であることを表す enable 信号が根に入力されていれば，発行の許可を表す grant 信号が出力される．この grant 信号は，request 信号を入力している子の arbiter cell のうち，1 番左にあるものへ enable 信号として入力される．同様にして，grant 信号は葉の arbiter cell まで到達する．grant 信号が葉の arbiter cell に到達したら，request 信号を入力している命令ウィンドウのエントリのうち，1 番左にあるものに向けて grant 信号が送出される．その grant 信号を受けて，命令ウィンドウのエントリから命令が発行される．

select 論理の遅延は，request 信号が葉から根へ伝搬する遅延，根での遅延，根から葉へ伝搬する遅延よりなる．従って，その遅延は arbiter cell の木の高さによって決まる．この木の高さは WS の関数になる．

また，本研究では，複数命令発行の場合の select 論理は **図 7** のようになるとしている．図は，WS=8，IW=2 の例である．クリティカル・パスは図の点線の部分である．図 7 より，select 論理の遅延時間は，IW にも依存している．よって，select 論理の遅延も WS と IW の関数になる．

表 1　プロセッサモデル

フェッチ/デコード/ 移動/発行/コミット幅	8 命令
ROB	4096 エントリ
LSQ	2048 エントリ
命令ウィンドウ	8x2, 16x1, 16x2, 32x1, 32x2, 64x1
実行レイテンシ	iALU 1, iMULT 3, iDIV 20 fpALU 2, fpMULT 4, fpDIV 12, fpSQRT 24
機能ユニット数	iALU 8, iMULT/DIV 8, Ld/St 8 fpALU 8, fpMULT/DIV/SQRT 8
分岐予測	gshare/2bc ハイブリッド (64K エントリ セレクタ, 履歴長 16 ビット/インデクス長 16 ビット gshare, 64K エントリ 2bc), 32 エントリ RAS, 4K エントリ/4 ウェイ BTB,
分岐予測ミスペナルティ	15+命令ウィンドウの段数
命令キャッシュ	64KB, 連想度 2, ライン幅 64B, ヒットレイテンシ 1 サイクル
データキャッシュ	64KB, 連想度 2, ライン幅 64B, ヒットレイテンシ 1 サイクル
2 次キャッシュ	2MB, 連想度 4, ライン幅 64B, ヒットレイテンシ 10 サイクル
メインメモリ	ファーストヒットレイテンシ 100 サイクル, バースト転送間隔 2 サイクル, バス幅 8B
命令 TLB	64 エントリ, 連想度 4, ブロックサイズ 4KB
データ TLB	128 エントリ, 連想度 4, ブロックサイズ 4KB
TLB ミスレイテンシ	120 サイクル

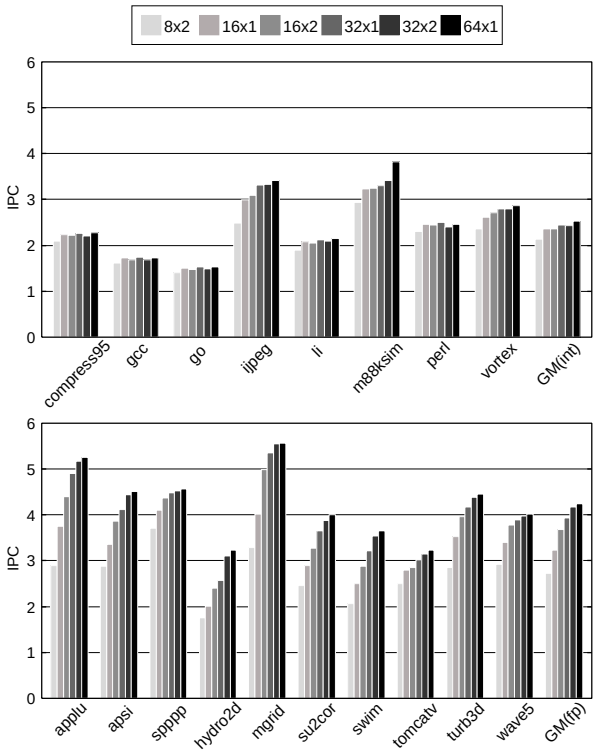


図 8　IPC

表 2　命令発行論理の遅延時間

WS	Wakeup (ps)	Select (ps)	合計 (ps)
8	160	820	980
16	182	820	1003
32	218	918	1136
64	265	918	1183

5. 評　　価

5.1 評価環境

IPC の評価には，SimpleScalar Tool Set Version 3.0[3] に含まれるスーパスカラ・プロセッサのシミュレータを修正したものを用いた．命令セットは SimpleScalar/PISA である．ベンチマーク・プログラムとして，SPEC CPU95 の 18 本を使用した．シミュレーション時間が過大にならないようにするために，命令ミックス，関数の出現頻度など，特徴をほぼ維持しつつ入力のパラメータを調節している．また，**表 1** に評価に用いたプロセッサ・モデルを示す．

遅延時間は，HSPICE を用いて測定した．トランジスタ・モデルには BPTM[5] の 0.10μm プロセスを用いた．また，配線抵抗と配線容量には文献 1) の値を用いた．

5.2 IPC の評価

図 8 に IPC の測定結果を示す．縦軸は IPC であり，横軸はベンチマークである．各ベンチマークごとに 6 本の棒グラフがあり，左から，8x2，16x1，16x2，32x1，32x2，64x1 の場合である．

図 8 より，命令ウィンドウのサイズの合計が大きい方が IPC が高いことがわかる．また，命令ウィンドウを分割すると IPC は下がる．これは，分割によって，機能ユニットへ発行できる命令が IWIN 内の命令だけに制限されるためである．compress95，gcc，go，li，perl では，16x1 よりも 16x2 の方が，あるいは 32x1 より 32x2 の方が IPC が低い．これは，命令ウィンドウを大きくする利益よりも，分割によって機能ユニットへ発行できる命令が制限される不利益の方が大きいためである．これらのベンチマークでは，命令ウィンドウが大きくなっても IPC があまり変化していない．つまり，命令ウィンドウが大きくなることによる利益が得られていない．また，int に比べて fp の方が命令ウィンドウが大きくなったときの IPC の増加が大きい．16x1 に対する 8x2 の IPC 低下率は，SPECint95 平均で 9.2%，SPECfp95 平均で 15.6%である．32x1 に対する 16x2 の IPC 低下率は，SPECint95 平均で 3.3%，SPECfp95 平均で 6.7%である．64x1 に対する 32x2 の IPC 低下率は，SPECint95 平均で 4.2%，SPECfp95 平均で 1.8%である．

5.3 遅延時間の評価

表 2 に命令発行論理の遅延時間を示す．表より，WS が増えると，遅延時間も増加していることがわかる．select 論理は，WS=8 から WS=16，WS=32 から WS=64 になっても遅延時間は変化していないが，これは select 論理を構成する木の高さがその範囲では変化しないためである．命令発行論理の遅延時間は，WS=8 から WS=16 では 2.2%増加している．同様に，WS=16 から WS=32 では 13.3%増加し，WS=32 から WS=64 では 4.1%増加している．

5.4 命令スループット

命令発行論理の遅延の削減によるクロック周波数の向上も

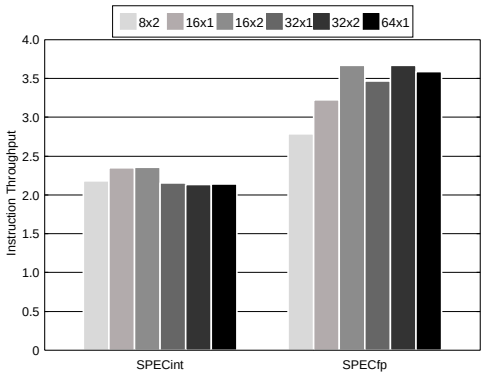


図 9 命令スループット

考慮した性能について評価する．ここでは，命令発行論理の遅延時間がクロック周波数を支配しているとする．また，性能を「命令スループット $= IPC \times$ クロック周波数」で定義する．

図 9 に命令スループットを示す．縦軸は命令スループットである．左の 6 本の棒グラフが SPECfp95 の平均であり，右の 6 本の棒グラフが SPECint95 の平均である．各グループごとに 6 本の棒グラフがあり，左から，8x2，16x1，16x2，32x1，32x2，64x1 の場合である．

図 9 より，int では 2 段に分割した方が性能が高くなるのは，32x1 から 16x2 になったときのみである．このとき，32x1 に対する 16x2 の性能向上は 9.6%である．一方，fp では 8x2 よりも 16x1 の方がスループットは高い．しかし，それ以上のサイズでは 2 段に分割した方が性能が高くなっている．32x1 に対する 16x2 の性能向上は，SPECfp で 5.8%である．64x1 に対する 32x2 の性能向上は，SPECfp で 2.2%である．int で命令ウィンドウが大きくなった場合に分割によって性能が向上していないのは，命令ウィンドウが大きくなることによる IPC の増加が少ないためである．

6. ま と め

命令ウィンドウを大きくすることによって，より多くの ILP を引き出すことができる．しかし，命令ウィンドウを大きくすると，命令発行論理の遅延が増加する．現在，命令発行論理の遅延はクリティカル・パスの一つとなっており，クロック周波数の向上を制限する要因となっている．

そこで，本論文では，2 段に分割した命令ウィンドウでスケジューリングを行う命令スケジューラを提案した．提案する命令スケジューラでは，1 段目の命令ウィンドウでデータ依存を考慮したプレスケジューリングを行い，2 段目の命令ウィンドウから命令を発行する．理想的には，実行可能となるまでの時間が短い命令だけが 2 段目の命令ウィンドウに存在するようになり，効率よく命令を発行することができる．そのため，命令ウィンドウの分割による IPC の低下を抑えることができる．提案する命令スケジューラにより，命令発行論理の遅延時間を短く保ったまま，命令ウィンドウを大きくすることができる．

評価の結果，32 エントリの命令ウィンドウを 2 段に分割した場合の IPC の低下率は，SPECint95 平均で 3.3%，SPECfp95 平均で 6.7%であった．64 エントリの命令ウィンドウを 2 段に分割した場合の IPC の低下率は，SPECint95 平均で 4.2%，SPECfp95 平均で 1.8%であった．しかし，命令発行論理の遅延の削減によるクロック周波数の向上も考慮すると，32 エントリの命令ウィンドウを 2 段に分割した場合には，SPECint95 平均で 9.6%，SPECfp95 平均では 5.8%命令スループットが向上し，64 エントリの命令ウィンドウを 2 段に分割した場合は SPECfp95 平均で 2.2%命令スループットが向上するという結果になった．

謝辞 本研究の一部は，文部科学省科学研究費補助金基盤研究（C）課題番号 15500036，文部科学省 21 世紀 COE プログラム，財団法人栢森情報科学振興財団研究助成の支援により行った．

参 考 文 献

1) V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, “Clock Rate versus IPC: The End of the Road for Conventional Microarchitecture” ISCA-27, pp. 248-259, Jun. 2000.
2) M. D. Brown, J. Stark, and Y. N. Patt, “Select-Free Instruction Scheduling Logic,” MICRO-34, pp. 204-213, Dec. 2001.
3) D. Burger and T. M. Austin, “The SimpleScalar Tool Set, Version 2.0,” Technical Report CR-TR-97-1342, Univ. of Wisconsin-Madison Computer Sciences Dept., Jun. 1997.
4) R. Canal and A. Gonzalez, “A Low-Complexity Issue Logic,” ISCA-14, pp. 327-335, May 2000.
5) Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, “New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design,” CICC 2000, pp. 201-204, Jun. 2000.
6) M. Goshima, K. Nishino, Y. Nakashima, S. Mori, T. Kitamura, and S. Tomita. “A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processor,” MICRO-34, pp. 225-236, Dec. 2001.
7) D. S. Henry, B. C. Kuszmaul, G. H. Loh, and R. Sami. “Circuits for Wide-Window Superscaler Processors,” ISCA-27, pp. 236-247, Jun. 2000.
8) P. Michaud and A. Seznec, “Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors,” HPCA-7, pp. 27-36, Jan. 2001.
9) S. Palacharla, N. P. Jouppi, and J. E. Smith, “Quantifying the Complexity of Superscalar Processors,” Technical Report CR-TR-96-1328, Univ. of Wisconsin-Madison, Nov. 1996.
10) S. Palacharla, N. P. Jouppi, and J. E. Smith, “Complexity-Effective Superscalar Processors,” ISCA-24, pp. 206-218, Jun. 1997.
11) S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, “A Scalable Instruction Queue Design Using Dependence Chains,” ISCA-29, pp. 318-329 May 2002.
12) J. Stark, M. D. Brown, and Y. N. Patt, “On Pipelining Dynamic Instruction Scheduling Logic,” MICRO-33, pp. 57-66, Dec. 2000.