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

2006－SLDM－124 (4)
2006／3／16

命令実行時の振る舞いに着目したローカル・スラック予測

小林良太郎† 林 久紘† 島田俊夫†

† 名古屋大学大学院工学研究科
〒 464-8603 名古屋市千種区不老町
E-mail: †{kobayasi,hayashi}@shimada.nuee.nagoya-u.ac.jp, ††shimada@.nuee.nagoya-u.ac.jp

あらまし 本稿では，発見的手法を用いてローカル・スラックを予測する機構を提案する．ローカル・スラックとは，他の命令に影響を与えることなく，その命令の実行レイテンシを増加させることのできるサイクル数である．提案機構は，分岐予測ミスやオペランド・フォワーディングなど命令実行時の振る舞いを観測しながら，予測するローカル・スラックを増減させ，実際のローカル・スラックに近づけていく．

キーワード スラック，高速化，低消費電力化

Local Slack Predictor Focusing on Dynamic Behavior of Instructions

Ryotaro KOBAYASHI†, Hisahiro HAYASHI†, and Toshio SHIMADA†

† Graduate School of Engineering, Nagoya University
Furo-Cho, Chikusa-Ku, Nagoya-City, Aichi, 464-8603 Japan
E-mail: †{kobayasi,hayashi}@shimada.nuee.nagoya-u.ac.jp, ††shimada@.nuee.nagoya-u.ac.jp

Abstract In this paper, we propose a mechanism that uses heuristics to predict local slack. Local slack of a dynamic instruction is the maximum number of cycles the execution latency of the instruction can be increased without affecting any other instruction. Our mechanism monitors the behavior of instructions (branch misprediction, operand forwarding, etc) and vary the predicted local slack for predicting the correct local slack.

Key words Slack, High Clock Frequency, Low Power

1. はじめに

近年，クリティカル・パスに関する情報を用いた，マイクロプロセッサの高速化や消費電力の削減に関する研究が数多く行われている [2], [3], [7], [9], [10]．クリティカル・パスとは，プログラム全体の実行時間を決定する動的な命令列で構成されるパスである．クリティカル・パス上の命令の実行レイテンシをたとえ 1 サイクルでも増加させると，プログラム全体の実行サイクル数が増加する．しかし，クリティカル・パス情報は命令がクリティカル・パス上にあるかないかの 2 通りしかなく，命令は 2 種類にしか分類できない．また，クリティカル・パス上の命令数は非クリティカル・パス上の命令数よりも大幅に少なく，それぞれのカテゴリごとに命令処理を分けた場合，負荷バランスが悪い．これらより，クリティカル・パス情報は，適用範囲が狭くなってしまう．

これに対し，クリティカル・パスの代わりに，命令の**スラック**を用いる手法が提案されている [4], [5]．命令のスラックとは，プログラム全体の実行サイクル数を増加させることなく，その命令の実行レイテンシを増加させることのできるサイクル数である．命令のスラックが分かれば，各命令がクリティカル・パス上にあるかどうかだけでなく，クリティカル・パス上にない命令の実行レイテンシを，プログラムの実行に影響しない範囲で，どの程度増加させられるのかが分かる．したがって，スラックを用いれば，命令を 3 種類以上のカテゴリに分けることができ，さらに，各カテゴリに属する命令数の不均衡を緩和することもできる．

各動的命令のスラックは，ある範囲を持った値である．スラックの最小値は常に 0 である．一方，スラックの最大値（**グローバル・スラック** [5]）は動的に決まる．スラックを最大限利用するためには，グローバル・スラックを求める必要がある．しかし，ある命令のグローバル・スラックを求めるためには，実行レイテンシの増加がプログラム全体の実行サイクル数に与える影響を，プログラムの実行中に調べなければならない．そのため，グローバル・スラックを求めるのは非常に難しい．

そこで，グローバル・スラックではなく，**ローカル・スラック** [5] を予測する手法が提案された [6], [8]．命令のローカル・スラックとは，プログラム全体の実行サイクル数だけでなく，後続命令の実行にも影響を与えないスラックの最大値である．ある命令のローカル・スラックは，依存関係のある後続命令に着目するだけで，容易に求めることができる．従来手法では，ある命令がレジスタ・データ，あるいは，メ

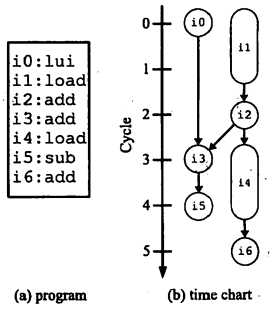

図 1 Slack

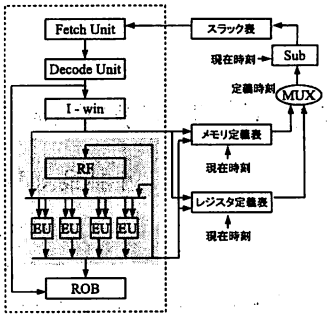

図 2 Conventional Scheme

モリ・データを定義した時刻と，そのデータを最初に参照した時刻の差から，当該命令のローカル・スラックを求め，それを基に，将来のローカル・スラックを予測する．

しかし，従来手法では，データを定義した時刻を保持するためのテーブルと，時刻の差を求めるための演算器を用意する必要がある．また，プログラムの実行と並列に，定義時刻を保持するテーブルの参照／更新や，時刻の引き算を行わなければならない．これらのコストが発生する原因は，データの定義／参照時刻を用いてローカル・スラックを直接計算することにある．

そこで，本論文では，発見的手法に基づいてローカル・スラックを予測する機構を提案する．この機構では，命令実行時の振る舞いを観測しながら，試行錯誤的にローカル・スラックを予測していく．これにより，ローカル・スラックを直接計算する必要がなくなる．さらに本論文では，応用例として，ローカル・スラックを用いた機能ユニットの低消費電力化手法を取り上げ，提案機構の効果について評価を行う．

2 章ではスラックについて説明する．3 章ではローカル・スラックを予測する従来手法を示す．4 章ではローカル・スラックを発見的に予測する手法と，提案手法を実現するための機構を示す．5 章では提案機構の評価を行う．最後に本論文をまとめる．

2. スラック

スラックの説明に用いるプログラムを図 1(a) に，その命令をプロセッサ上で実行する過程を図 1(b) に示す．図において，ノードは命令を示し，エッジは命令間のデータ依存関係を示す．縦軸は命令を実行したサイクルを示す．ノードの長さは命令の実行レイテンシを示す．実行レイテンシは，命令 i1 と i4 が 2 サイクル，その他が 1 サイクルである．

ここで，i0 のスラックについて考える．i0 の実行レイテンシを 3 サイクル増加させた場合，それに直接的，間接的に依存する i3，i5 の実行が遅れる．その結果，i5 は，プログラム中もっとも最後に実行される i6 と同時刻に実行される．したがって，i0 の実行レイテンシをこれ以上増加させると，プログラム全体の実行サイクル数が増加する．つまり，i0 のグローバル・スラックは 3 である．このように，ある命令のグローバル・スラックを求めるためには，その命令の実行レイテンシの増加が，プログラム全体の実行に与える影響を調べる必要がある．そのため，グローバル・スラックの判定は非常に難しい．

一方，i0 の実行を 2 サイクル増加させた場合，後続命令の実行に影響は与えない．しかし，これ以上実行レイテンシを増加させると，直接的，間接的に依存関係にある i3 と i5 の実行が遅れる．つまり，i0 のローカル・スラックは 2 である．このように，ある命令のローカル・スラックを求めるには，その命令に依存する命令への影響に着目すれば良い．したがって，ローカル・スラックは比較的容易に判定することができる．

3. 従来手法

例えば図 1(b) の i0 がデータを定義した時刻 0 と，そのデータが i3 によって最初に参照された時刻 3 との差から，さらに 1 を引き，i0 のローカル・スラックは 2 であると計算する．そして，それを基に，i0 を次に実行する場合のローカル・スラックは 2 であると予測する．

図 2 に，従来手法の概略を示す．図において，点線で囲われた部分はプロセッサを示す．プロセッサは，フェッチ・ユニット，デコード・ユニット，命令ウインドウ (I-win)，レジスタ・ファイル (RF)，実行ユニット (EU)，リオーダ・バッファ (ROB) を持つ．プロセッサの右側は，従来のローカル・スラック予測機構を示す．ローカル・スラック予測機構は，レジスタ・データ，および，メモリ・データを定義した時刻を保持するための定義表と，時刻の差を求めるための演算器を持つ．さらに，各命令のローカル・スラックを保持するためのスラック表を持つ．

図 1(b) の i0 のローカル・スラックを例に，従来機構の動作を簡単に説明する．i0 はデータを定義する時に，i0 自身と共に現在時刻 0 を定義表に記録する．i3 はデータを使用する時に，データを定義した命令 i0 とデータを定義した時刻（定義時刻）0 を，定義表から得る．そして，現在時刻 3 と定義時刻 0 との差分からさらに 1 を引くことで，i0 のローカル・スラック 2 を求める．求めたスラックは，スラック表の i0 に対応するエントリに記録する．i0 を次にフェッチしたときに，スラック表を参照し，得られたスラックから，i0 のローカル・スラックは 2 であると予測する．

以上のように，従来手法では，定義表と演算器を用意する必要があり，ハードウェア・コストが増大する．また，プログラムの実行と並列に，定義表の参照／更新と時刻の引き算を行わなければならないため，高速な動作を必要とし，それが消費電力に大きな影響を及ぼす可能性がある．こうした問題が発生する原因は，データの定義／参照時刻に着目してローカル・スラックを直接計算することにある．

4. ローカル・スラックを発見的に予測する手法

従来手法に対し，本論文では，ローカル・スラックを発見的に予測する手法を提案する．この手法では，命令実行時の振る舞いを観測しながら，予測するローカル・スラック（**予測スラック**）を増減させ，予測スラックを実際のローカル・スラック（**ターゲット・スラック**）に近づけていく．試行錯誤的に予測を行うため，従来手法のようにローカル・スラックを直接計算する必要はない．

以下では，説明を簡単にするために，まず，提案手法の基

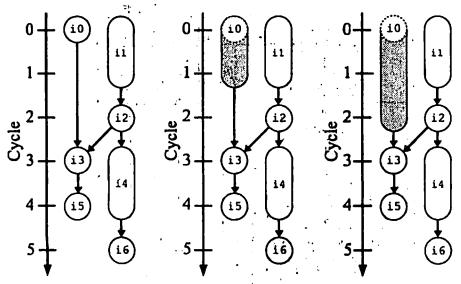

図 3 Basic Operation

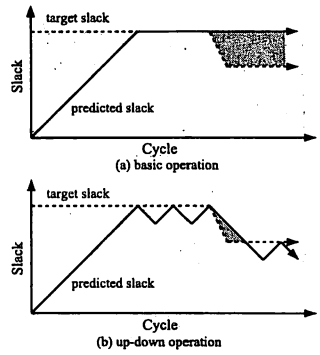

図 4 Target Slack Reduction

本的な動作を説明する．その後，ターゲット・スラックの動的な変化に対応するための修正を加える．最後に提案手法の構成について説明する．

4.1 基本動作

まず，提案手法の基本動作を示す．命令フェッチ時にローカル・スラックを予測し，予測スラックに基づいて命令の実行レイテンシを増加させる．どの命令に対しても，それを初めてフェッチするときには，ローカル・スラックは 0 であると予測する．つまり，予測スラックの初期値は 0 とする．その後は，命令実行時の振る舞いを観測しながら，予測スラックを，ターゲット・スラックに到達するまで，徐々に増加させていく．

つぎに，基本動作において，命令実行時の振る舞いを基に，予測スラックがターゲット・スラックに到達したかどうかを判定する方法を説明する．ここで，ある命令の予測スラックを増加させていき，その値がターゲットスラックに到達したという状況を考える．このとき，当該命令は，実行レイテンシを 1 サイクルでも増加させると，それに依存する命令の実行を遅れさせてしまう状態にある．したがって，この状態にある命令は，以下のいずれかの振る舞いを見せると考えられる．

- 分岐予測ミス
- キャッシュ・ミス
- 後続命令に対するオペランド・フォワーディング
- 後続命令に対するストア・データ・フォワーディング

そこで，提案手法では，命令実行時の振る舞いが，上記のいずれかに該当した場合，予測スラックはターゲット・スラックに到達したと判定し，そうでない場合，まだ到達していないと判定する．上記をこれ以降**ターゲット・スラック到達条件**と呼ぶこととする．これらの条件はプロセッサ上で容易に観測することができる．

提案手法の基本動作に基づいて，図 1(a) のプログラムを繰り返し実行する過程を図 3 に示す．図 3(a)〜(b) は，それぞれ 1〜3 回目の実行を示す．図において，ノードの網掛け部分は，予測スラックに応じて増加させた実行レイテンシを示す．図では，説明を簡単にするため，i0 のローカル・スラックのみを予測の対象とし，予測スラックは 1 回につき 1 ずつ増加させるとする．

1 回目の実行では，i0 の予測スラックは 0 である．この場合，i0 の実行時の振る舞いは，ターゲット・スラック到達条件のいずれにも該当しないので，予測スラックはターゲット・スラックにまだ到達していない．そこで，i0 の予測スラックを 1 増加させる．その結果，2 回目の実行では，i0 の予測スラックは 1 となる．この場合も，予測スラックはターゲット・スラックに到達していない．そこで，i0 の予測スラックをさらに 1 増加させる．これにより，3 回目の実行では，i0 の予測スラックは 2 となる．その結果，i0 は後続命令に対しオペランド・フォワーディングを行う．これにより，ターゲット・スラック到達条件を満たす．予測スラックはターゲット・スラックに到達したので，これ以上増加させない．以上のようにして，i0 のローカルスラックを予測する．

4.2 ターゲット・スラックの動的な変化への対応

基本動作では，ターゲット・スラックの動的な変化に十分に対応することができない．ターゲット・スラックが動的に変化しても，それが予測スラックよりも大きいのであれば，予測スラックは新たなターゲット・スラックを目指して増加するだけなので，問題はない．しかし，ターゲット・スラックが予測スラックより小さくなると，予測スラックは変化することなく，そのままの値を維持するので，ターゲット・スラックを上回った分（スラック予測ミスペナルティ）だけ，後続命令の実行を遅れさせてしまう．これが，性能に悪影響を及ぼす可能性がある．

この問題に対し，まず，ターゲット・スラックが予測スラックよりも小さくなったら，予測スラックを減らすという解決手法を提案する．しかし，ターゲット・スラックが急速に増減を繰り返す場合，この手法を導入しても，予測スラックをターゲット・スラックに追従させることはできない．その結果，ターゲット・スラックが予測スラックよりも小さくなるという状況が頻繁に発生する．そこでさらに，信頼性を導入して，予測スラックの増加は慎重に行い，予測スラックの減少は迅速に行うという解決手法を提案する．

以下では，上記 2 つの解決方法について詳しく説明する．

4.2.1 予測スラックの減少

予測スラックの減少を実現する方法として，スラック予測を行わなかった場合の後続命令の実行時刻（後続命令の本来実行されるべき時刻）を利用するという方法が考えられる．後続命令の本来実行されるべき時刻が分かれば，スラック予測のミスにより後続命令の実行時刻が遅れたかどうかを調べることができる．あるいは，ターゲット・スラックを直接計算し、予測スラックと比較することもできる．しかし，いずれにしても，命令の実行時刻を決定しうる様々な要素（資源制約，データ依存，制御依存など）を考慮して，後続命令の本来実行されるべき時刻を計算しなければならないので，簡単に実現することはできない．

そこで我々は，先程述べた，ターゲット・スラック到着条件に着目する．この条件を用いれば，予測スラックがターゲット・スラックを下回っていることと，ターゲット・スラックに到着したことが容易に分かる．この特徴を利用し，予測スラックがターゲット・スラックに到着したら，その後は逆に，ターゲット・スラックを下回るまで予測スラックを減少

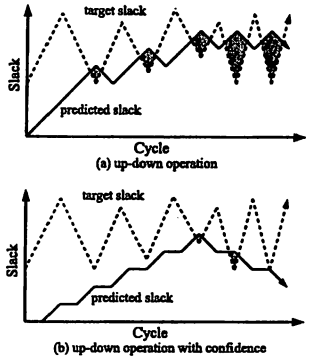

図 5 Rapid Change of Target Slack

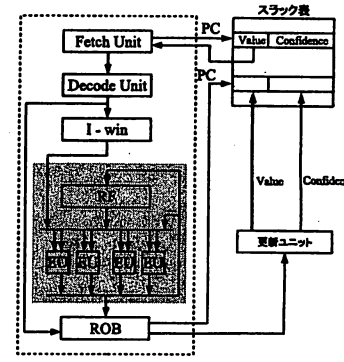

図 6 Our Proposal

させることとする．こうすることにより，非常に簡単な修正で，ターゲット・スラックの動的な減少に対応できるようになる．ターゲット・スラックを下回る分の予測スラックが無駄になるが，十分に許容できると考える．

図 4 を用いて，基本動作の問題点と，その解決手法について説明する．図は，ターゲット・スラックが動的に減少した場合に，予測スラックがどのように変化するのかを示す例である．図において縦軸はスラックを示し，横軸は時刻を示す．折れ線グラフは，点線がターゲット・スラックの場合，実線が予測スラックの場合である．網掛け部分は，予測スラックがターゲット・スラックを超えてしまう箇所を示す．図 4(a) は，基本動作の場合，(b) は，本項で提案する解決手法を導入した場合である．

図 4(a) において，予測スラックはターゲット・スラックに到達するまで増加していく．その後，ターゲット・スラックが減少し，予測スラックより小さくなる．しかし，予測スラックはそのままの値を維持し，後続命令の実行を継続的に遅れさせてしまう．

一方，図 4(b) に示すように，修正後の動作では，まず，予測スラックはターゲット・スラックに到達するまで増加していく．到着後，予測スラックは減少するが，ターゲット・スラックを下回るので，即座に増加に転じ，再びターゲット・スラックに到達する．この変化を，しばらくの間繰り返す．その後，ターゲット・スラックが減少すると，予測スラックは，ターゲット・スラックを下回るまで減少していき，再び増減を繰り返す．こうして，ターゲット・スラックの減少にあわせて予測スラックを減らすことができるようになる．

4.2.2 信頼性の導入

ターゲット・スラックの急速な変化に対応するため，基本動作をさらに修正する．まず，予測スラックごとに，ターゲット・スラック到達条件が信頼できるかどうか示すカウンタ（**信頼性カウンタ**）を導入する．カウンタ値は，命令がターゲット・スラック到達条件を満たしていれば減少させ，そうでなければ増加させる．そして，カウンタ値が 0 になったら予測スラックを減少させ，カウンタ値がある閾値を超えたら予測スラックを増加させる．

予測スラックの増加を慎重に行うため，予測スラックを増加させる際に，カウンタ値を 0 にリセットすることとする．また，予測スラックの減少を迅速に行うため，命令がターゲット・スラック到達条件を満たしていれば，カウンタ値を 0 にリセットすることとする．

図 5 を用いて，前項で示した解決手法の問題点と，それを解決するための手法について説明する．図は，ターゲット・スラックが急速に増減を繰り返した場合に，予測スラックがどのように変化するのかを示す例である．図 5(a) は，基本動作に予測スラックの減少を導入した場合，(b) は，さらに信頼性も導入した場合である．

図 5(a) において，予測スラックはターゲット・スラックを目指して変化しようとするが，急速な変化に追従できず，頻繁にターゲット・スラックを超えてしまうことが分かる．

一方，図 5(b) に示すように，信頼性を導入すると，予測スラックはターゲット・スラックを目指して緩やかに増加していき，ターゲット・スラックに到達する（あるいはそれを超える）と即座に減少するという変化を繰り返す．これにより，予測スラックがターゲット・スラックを超える頻度を下げることができる．

4.3 ハードウェア構成

図 6 に，提案手法の構成を示す．図において，点線で囲われた部分はプロセッサを示す．プロセッサの右側は，我々の提案するローカル・スラック予測機構を示す．提案機構は，予測スラックを保持するためのスラック表と，スラック表に更新する値を計算する更新ユニットで構成される．

スラック表は，命令の PC をインデクスとし，各エントリは対応する命令の予測スラック（Value）と，ターゲット・スラック到達条件の信頼性（Confidence）を保持する．ROB の各エントリにフィールドを追加し，Confidence，Value，ターゲット・スラック到達条件（4.1 節を参照）を満たしたかどうかを示す R フラグ（reach フラグの略）を保持する．更新ユニットの構成は後程述べる．

プロセッサは，フェッチ時に，命令の PC をインデクスとしてスラック表を参照し，対応するエントリから予測スラックを得る．実行時に，命令の振る舞いを観測し，ターゲット・スラック到達条件を満たすのであれば，対応する ROB エントリの R フラグを 1 にセットする．そしてコミット時に，更新ユニットを用いて，スラック表の対応するエントリを更新する．

スラック表の更新に関係するパラメータとその内容を以下に示す．

- $Vmax$: 予測スラックの最大値
- $Vmin$: 予測スラックの最小値（$= 0$）
- $Vinc$: 予測スラックの 1 回あたりの増加量
- $Vdec$: 予測スラックの 1 回あたりの減少量
- $Cmax$: 信頼性の最大値
- $Cmin$: 信頼性の最小値（$= 0$）
- Cth: 信頼性の閾値
- $Cinc$: 信頼性の 1 回あたりの増加量
- $Cdec$: 信頼性の 1 回あたりの減少量

$Vmin = 0$，$Cmin = 0$ である．4.2 節において，ターゲッ

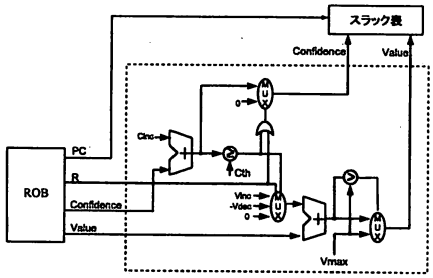

図 7 更新ユニット

ト・スラック到達条件が成立すれば信頼性を 0 にリセットするとしたので，$Cdec = Cth$ である．また，予測スラックを増加させる際に，信頼性を 0 にリセットするとしたので，$Cmax = Cth$ である．

図 7 に，更新ユニットの回路構成を示す．図において，点線で囲われた部分は更新ユニットを示す．更新ユニットは，信頼性や予測スラックを増減させるための加算器，加算器の出力結果をパラメータと比較するための比較器で主に構成される．

更新ユニットは，R フラグが 1 ならば，つまり，ターゲット・スラック到達条件が成立していれば，信頼性を 0 にリセットし，そうでなければ $Cinc$ 増加させる．信頼性が Cth に達したら，予測スラックを $Vinc$ 増加させ，信頼性を 0 にリセットする．信頼性が 0 になったら，予測スラックを $Vdec$ 減少させる．上記のいずれでもなければ，予測スラックは変化させない．増加させた予測スラックが $Vmax$ を超えたら，予測スラックを $Vmax$ に戻す．

図から分かるように，更新ユニットでは，2 つの加算器と 2 つの比較器が直列に接続されている．扱うデータのサイズによっては，回路の遅延時間がサイクル時間に悪影響を与える可能性がある．

我々はこの問題を解決するために，更新ユニットの高速化を行った．高速化のため，加算器のキャリー出力を利用して比較器を取り除き，加算器を多重化して加算処理を並列化した．

図 8 に，高速化した更新ユニットの回路構成を示す．まず，高速化した更新ユニットでは，Cth を 2 の乗数，$Vmax$ を 2 の乗数から 1 引いた値とし，加算器のキャリー出力を比較器の出力として用いている．Cth は 2 の乗数なので，信頼性が Cth に達したときに，信頼性に $Cinc$ を加える加算器のキャリー出力が 1 となる．また，$Vmax$ は 2 の乗数から 1 引いた値なので，予測スラックが $Vmax$ を超えたときに，予測スラックを $Vinc$ 増加させる加算器のキャリー出力が 1 となる．したがって，比較器がなくても，更新値を計算することができる．

つぎに，予測スラックを増減させる加算器を多重化し，一方で $Vdec$ との加算を，もう一方で $-Vdec$ との加算を行っている．これにより，加算器の数は 1 つ増加するが，信頼性の加算と並列に，予測スラックの加算を行うことができるようになる．

5. スラック予測機構の評価

本章では，まず評価環境について述べる．次に，評価モデル，評価内容について述べ，最後に提案機構を評価した結果を示す．

5.1 評価環境

シミュレータには，SimpleScalar Tool Set [1] のスーパースカラ・プロセッサ用シミュレータを用い，提案方式を組み込んで評価した．命令セットには MIPS R10000 を拡張した SimpleScalar/PISA を用いた．ベンチマーク・プログラムは，SPECint2000 の bzip2, gcc, gzip, mcf, paser, perl, votex, vpr の 8 本を使用した．gcc では 1G 命令，その他では 2G 命令をスキップした後，100M 命令を実行した．測定条件として表 1 を用いた．スラック表のエントリ数は従来方式 [8] と同一とした．また，パラメータの組合せの数は膨大になるので，$Vinc = Cinc = 1$，$Vdec = Vmax$ に固定し，$Vmax$ と Cth だけを変化させて評価を行った．

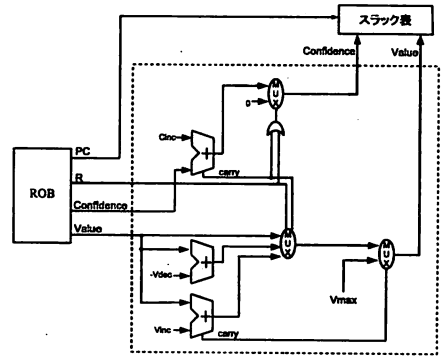

図 8 高速化した更新ユニット

表 1 測定条件

フェッチ幅	8 命令
発行幅	8 命令
命令ウィンドウ	128 エントリ
ROB	256 エントリ
LSQ	64 エントリ
機能ユニット数	iALU 6, iMULT/DIV 1, fpALU 1, fpMULT/DIV/SQRT 1
命令キャッシュ	完全，ヒットレイテンシ 1 サイクル
データキャッシュ	32KB, 2 ウェイ，32B ライン，4 ポート，ミスペナルティ6 サイクル
2 次キャッシュ	2MB, 4 ウェイ，64B ライン，ミスペナルティ36 サイクル
ストアセット	8K エントリ SSIT, 4K エントリ LFST
分岐予測機構	BTB 2048 エントリ，4 ウェイ，gshare 6 ビット履歴 8K エントリ PHT，RAS(Return Address Stack) 16 エントリ，分岐予測ミスペナルティ5 サイクル
スラック表	8192 エントリ，2 ウェイ

以下モデルについて評価した．

- 基本モデル: 予測スラックの増加のみを行うモデル
- 増減モデル: 予測スラックの増減を行うモデル
- 慎重型増減モデル: 増減モデルに信頼性を導入したモデル
- 従来モデル: 時刻の差からスラックを求め，それを基に，将来のスラックを予測するモデル

最初の 3 つが提案手法に基づくモデル，最後の 1 つが従来手法に基づくモデルである．

評価では，**スラック命令**の数，予測スラックの平均，IPC (性能) を測定する．スラック命令とは，予測スラックに基づいて実行レイテンシを 1 サイクル以上増加させた命令で

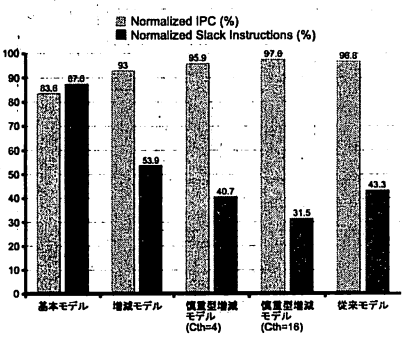

図 9 Slack Instructions vs. IPC (Vmax=1)

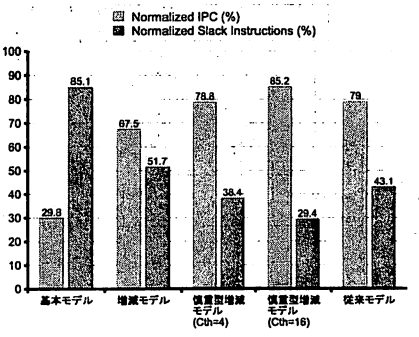

図 10 Slack Instructions vs. IPC (Vmax=15)

ある.

5.2 評価結果

図 9 と図 10 に，予測スラックの最大値 ($Vmax$) を，それぞれ 1，15 として，各モデルの IPC とスラック命令数を測定した結果を示す．縦軸は，スラック予測を行わない場合で正規化した IPC，および，全実行命令数で正規化したスラック命令数を，ベンチマーク平均で示す．慎重型増減モデルの評価結果は信頼性の閾値によって変化するので，信頼性の閾値 (Cth) が 4 の場合と 16 の場合を示す．2 本で組になっている棒グラフは，左側が IPC の場合，右側がスラック命令数の場合である．

図 9 より，提案モデルの性能は，基本モデル，増減モデル，慎重型増減モデルの順に高くなり，慎重型増減モデルにおいては，Cth が大きいほど性能が高いことが分かる．逆に，提案モデルのスラック命令数は，性能が高いほど少なくなることが分かる．このように，性能とスラック命令数はトレードオフの関係にあるので，どのモデルを用いるかは，応用によって異なる．例えば，性能を重視するのであれば，$Cth = 16$ の慎重型増減モデルがもっとも良い．このとき，2.4%の IPC 低下で，31.5%もの命令について実行レイテンシを 1 サイクル増加させることができる．

図 9 より，慎重型増減モデル ($Cth = 4$) を従来モデルと比較すると，IPC が 0.9% ポイント，スラック命令数が 2.6%ポイント低下するものの，ほぼ同等の結果を示している．このことから，提案手法は，スラックを直接計算しないにも関わらず，従来手法と同じような結果が得られることが分かる．

図 10 より，$Vmax$ を 1 から 15 に増加させた場合でも，図 9 と似た傾向を示すが，どのモデルにおいても，性能とスラック命令数が減少していることが分かる．これは，$Vmax$ を増加させることで，予測ミス・ペナルティが増加することが原因である．また，提案モデルでは，モデル間の性能差が非常に大きくなっている．このことから，$Vmax$ が大きいほど，性能低下を抑制する効果が高くなることが分かる．

図 10 より，慎重型増減モデル ($Cth = 4$) を従来モデルと比較すると，IPC が 0.2% ポイント，スラック命令数が 4.7%ポイント減少している．しかし，増加させることのできた実行レイテンシを測定した結果，慎重型増減モデル ($Cth = 4$) の場合平均 8.9 サイクル，従来モデルの場合平均 5.9 サイクルとなることが分かった．また，増加させることのできた実行レイテンシの合計は，慎重型増減モデル ($Cth = 4$) の方が従来モデルよりも約 1.3 倍多いことが分かった．

以上のように，提案手法はより少ないハードウェアコストで，従来手法と同様かそれ以上の結果を示すことが分かった．

6. まとめ

今回我々は発見的手法によってスラックを予測する機構を提案した．命令の振る舞いから間接的にスラックを予測するため，従来手法に対しより単純なハードウェアで実現できる．評価の結果，スラック表の信頼性の閾値が 16 の場合，わずか 2.4%の IPC 低下で，31.5%の命令について実行レイテンシを 1 サイクル増加させることができると分かった．

謝辞 本研究の一部は，株式会社半導体理工学センター，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 プログラムの支援により行った．

文献

[1] D. Burger, et al., "The Simplescalar Tool Set Version 2.0," Technical Report 1342, Department of Computer Sciences, University of Wisconsin-Madison , June 1997.

[2] 千代延昭宏ほか：低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案，情報処理学会研究報告 2002-ARC-149, 2002 年 8 月.

[3] B. Fields, et al., "Focusing Processor Policies via Critical-Path Prediction," In *Proc. ISCA-28*, June 2001.

[4] B. Fields, et al., "Using Interaction Costs for Microarchitectural Bottleneck Analysis," In *Proc. MICRO-36*, Dec. 2003.

[5] B. Fields, et al., "Slack: Maximizing Performance under Technological Constraints," In *Proc. ISCA-29*, May 2002.

[6] 福山 智久 ほか：スラック予測を用いた省電力アーキテクチャ向け命令スケジューリング，先進的計算基盤システムシンポジウム SACSIS2005, 2005 年 5 月.

[7] 小林良太郎ほか：データフロー・グラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構，2001 年並列処理シンポジウム JSPP2001, 2001 年 6 月.

[8] 劉 小路 ほか：クリティカリティ予測のためのスラック予測，先進的計算基盤システムシンポジウム SACSIS2004, 2004 年 5 月.

[9] J. S. Seng, et al., "Reducing Power with Dynamic Critical Path Information," In *Proc. MICRO-34*, Dec. 2001.

[10] E. Tune, et al., "Dynamic Prediction of Critical Path Instructions," In *Proc. HPCA-7*, jan. 2001.