発見的手法に基づくローカル・スラック予測機構

小林 良太郎[†] 林 久 紘[†,☆] 島田 俊夫[†]

本論文では，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．評価した結果，わずか 2.5%の IPC 低下で，31.6%の命令の実行レイテンシを 1 サイクル増加させることができることが分かった．

Local Slack Predictor Based on Heuristics

Ryotaro Kobayashi,[†] Hisahiro Hayashi[†,☆] and Toshio Shimada[†]

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. Our evaluation results show that the execution latency of 31.6% instructions can be increased bye one cycle with only a 2.5% performance degradation.

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),8),11),13)]．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つねに 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),10)]．命令のローカル・スラックとは，プログラム全体の実行サ

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
☆ 現在，株式会社日立製作所
Presently with Hitachi, Ltd.

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．6 章ではスラック予測機構のハードウェアに関する評価を行う．7 章では提案機構を消費電力の削減に応用した場合の評価を行う．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，プログラム中最も後に実行される `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

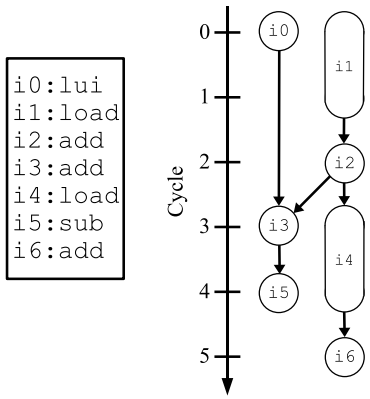

図 1 スラック
Fig. 1 Slack.

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よい．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

たとえば図 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

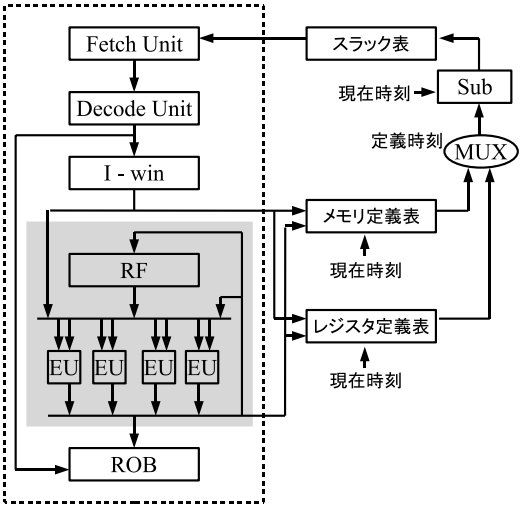

図 2 従来機構
Fig. 2 Conventional scheme.

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ときに，i0 自身とともに現在時刻 0 を定義表に記録する．i3 はデータを使用するときに，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．

ここでは概略のみを示したが，従来手法は，ローカル・スラックを計算するだけではなく，命令の実行を遅れさせたか否かという情報に基づいて，ローカル・スラックの動的な変化に対応する機能も備えている[6]．

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．

次に，基本動作において，命令実行時の振舞いを基に，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．また，命令間の依存関係として，制御依存，キャッシュ・ラインを介した依存，レジスタ・データ依存，メモリ・データ依存をあげることができる．したがって，予測スラックがターゲット・スラックに到達した命令は，以下のいずれかの振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**

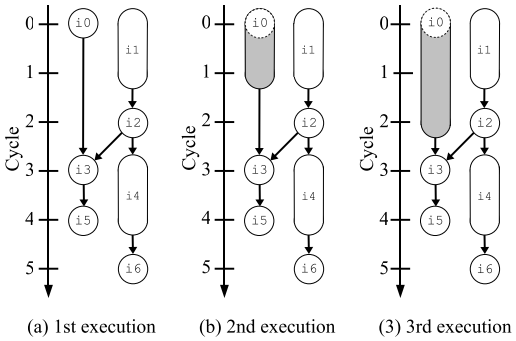

図 3 基本動作
Fig. 3 Basic operation.

ゲット・スラック到達条件と呼ぶこととする．

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，先ほど述べた，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，

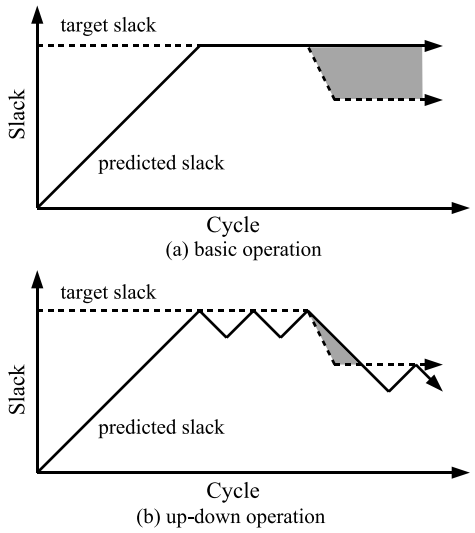

図 4 ターゲット・スラックの減少
Fig. 4 Target slack reduction.

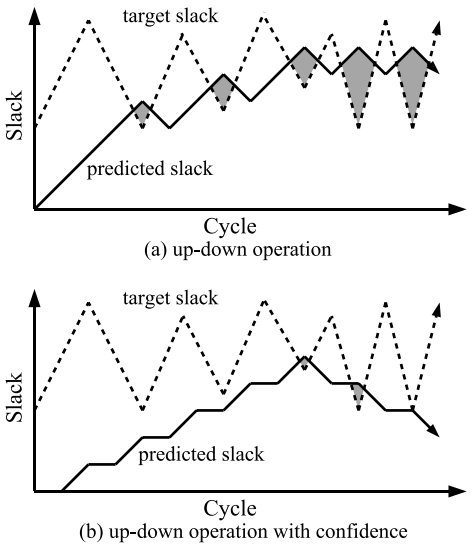

図 5 ターゲット・スラックの急速な変化
Fig. 5 Rapid change of target slack.

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以上になっ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

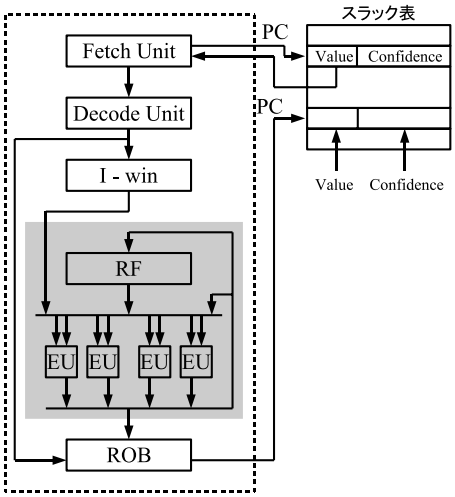

図 6 提案機構
Fig. 6 Our proposal.

の信頼性（Confidence）を保持する．

命令は，フェッチ時に，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．そして，コミット時に，命令実行時の振舞いを基にしてスラック表を更新する．

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．ただし，$Vmin = 0$，$Cmin = 0$ である．

- $Vmax$: 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

スラック表の更新の流れを以下に示す．ターゲット・スラック到達条件（4.1 節を参照）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 以上になったら，予測スラックを $Vinc$ 増加させ，信頼性を 0 にリセットする．一方，信頼性が 0 になったら，予測スラックを $Vdec$ 減少させる．

なお，4.2 節において，ターゲット・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．さらに，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．

5. スラック予測機構の評価

本章では，まず評価モデル，評価環境について述べる．次に，評価結果について述べる．

5.1 評価モデル

以下のモデルについて評価した．

- **NO-DELAY モデル**
  予測スラックに基づいた実行レイテンシの増加を行わないモデルである．
- **B モデル**
  提案手法の基本動作のみ行うモデルである．
- **BCn モデル**
  提案手法の基本動作に，信頼性を導入したモデルである．モデルに付加した数値 n は，信頼性の閾値 Cth を表す．
- **BD モデル**
  提案手法の基本動作に，予測スラックの減少を導入したモデルである．
- **BDCn モデル**
  提案手法の基本動作に，予測スラックの減少と信頼性を導入したモデルである．モデルに付加した数値 n は，信頼性の閾値 Cth を表す．

B モデル，BCn モデル，BD モデル，BDCn モデルは，提案方式を基にしたモデルであるため，これらを提案モデルと呼ぶこととする．

5.2 評 価 環 境

シミュレータには，SimpleScalar Tool Set [1] のスーパスカラ・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 bzip，gcc，gzip，mcf，paser，perlbmk，votex，vpr の 8 本を使用した．gcc では 1 G 命令，その他では 2 G 命令をスキップした後，100 M 命令を実行した．測定条件として**表 1** を用いた．従来方式との比較のため，スラック表のエントリ数は，従来方式[10] と同一とした．

スラック表の更新に関するパラメータおいて，変化させうるものは，$Vmax$，$Vinc$，$Vdec$，Cth，$Cinc$ である．これらの組合せは膨大な数になるので，いくつかのパラメータをある値に固定する．まず，$Cinc$ と Cth の比はスラックを増加させる頻度を表すので，$Cinc = 1$ に固定し，Cth だけを変化させることとする．次に，予測スラックをできるだけターゲット・スラックに近づけるために，$Vinc = 1$ に固定する．最後に，予測スラックをできるだけ早く減少させるため

表 1 測定条件
Table 1 Processor configurations.

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	32 KB, 2 ウェイ，32 B 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	2 MB, 4 ウェイ，64 B ライン，ミスペナルティ 36 サイクル
ストアセット	8 K エントリ SSIT, 4 K 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 8 K 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，($Vmax + Cth$) ビットライン

に，$Vdec = Vmax$ に固定する．以上より，本章では，$Vmax$ と Cth だけを変化させて，提案方式の評価を行うこととする．ただし，比較を容易にするため，Cth は，5，15 の 2 通りに，$Vmax$ は，1，5，15 の 3 通りに制限する．

5.3 スラック予測精度

ここではまず，各動的実行命令に対し，実際のスラック（**実スラック**）を測定する．具体的には，NO-DELAY モデルにおいて，ある命令がレジスタ・データ，あるいは，メモリ・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・スラックを求める．そのため，データを定義しない命令（分岐命令）のスラックは無限大となる．

図 7 に，実スラックの累積分布を示す．縦軸は，全実行命令数に占める割合を示し，横軸は実スラックを示す．折れ線グラフは，実線がベンチマーク平均を示し，点線が各ベンチマークを示す．実スラックが 32 サイクルの点において，上から順に vpr，bzip，gzip，parser，平均，perlbmk，gcc，vortex，mcf の場合である．

図が示すとおり，実スラック 0 の命令は平均 52.7%存在する．実スラックが増えるにつれて，実行命令数に占める割合は徐々に飽和していく．また，実スラックが，30 以上存在する命令は平均 28.9%存在することが分かる．しかし，通常のプロセッサで，命令の実行レイテンシを数十サイクル以上増加させると，それらの命令がプロセッサ内のバッファ（ROB，I-win 等）を占有するため，性能が大幅に低下する[10]．こうした大きなスラックをどのように利用するのかは，今のところ十分には検討されていない．

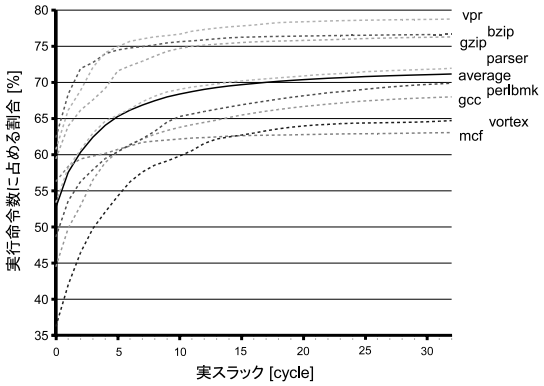

図 7 実スラック
Fig. 7 Real slack.

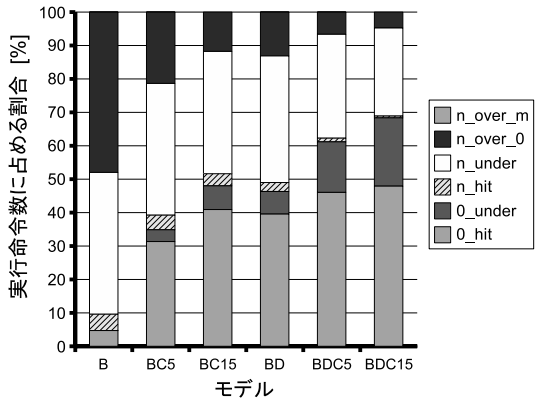

図 8 スラック予測精度（$Vmax = 1$）
Fig. 8 Slack prediction accuracy ($Vmax = 1$).

図 8，**図 9**，**図 10** に，$Vmax$ を，それぞれ，1，5，15 とした場合において，各提案モデルのスラック予測精度を測定した結果をベンチマーク平均で示す．図の縦軸は，全実行命令数に占める割合を示し，横軸はモデルを示す．棒グラフは，6 つの部分からなり，上の 4 つが，スラックを n（n は 1 以上）と予測した場合，下の 2 つが，スラックを 0 と予測した場合である．スラックを n と予測した場合，上から順に，n が実スラック m（m は 1 以上）を超えた場合，実スラック 0 を超えた場合，実スラックを下回った場合，実スラックと一致した場合である．一方，スラックを 0 と予測した場合，上から順に，実スラックを下回った場合，実スラックと一致した場合である．ただし，$Vmax = 1$ の場合，予測スラック n が実スラック m を上回る場合はないので，棒グラフは 5 つの部分からなる．これ以降，予測スラックと実スラックが一致することを，予測がヒットすると呼ぶ．

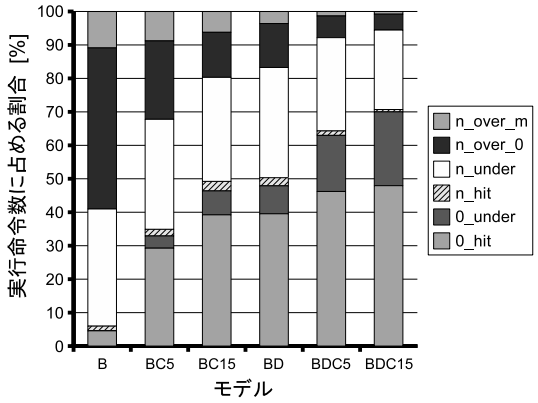

図 9 スラック予測精度（$Vmax = 5$）
Fig. 9 Slack prediction accuracy ($Vmax = 5$).

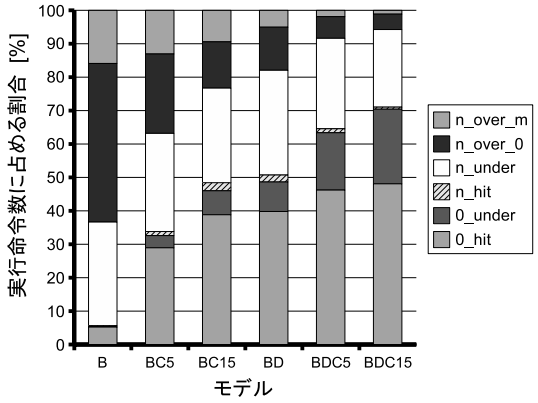

図 10 スラック予測精度（$Vmax = 15$）
Fig. 10 Slack prediction accuracy ($Vmax = 15$).

図より，予測がヒットする割合は B モデルが最も低いことが分かる．これに対し，予測スラックの減少を導入したモデル（BD モデル）と，信頼性を導入したモデル（BCn モデル）は，どちらもヒット率向上に効果があることが分かる．また，両者をともに導入したモデル（BDCn モデル）は，さらに高い効果が得られる．信頼性を導入したモデルの場合，信頼性の閾値（モデルに付加した数字）が高いほど，ヒット率は高くなる．なお，予測がヒットするのは，$Vmax = 1$ の B モデルを除くと，実スラックが 0 の場合がほとんどである．この場合，スラックは利用できない．

予測スラックが実スラックを上回る場合，実スラックを超えるスラックを利用してしまうことになる．したがって，予測ミスによるペナルティが発生する．図より，予測ミスペナルティの発生率は，ヒット率が高いほど下げることができることが分かる．一方，予測スラックが実スラックを下回る場合，予測ミスペナルティは発生しない．この場合，予測スラックが 1 以上

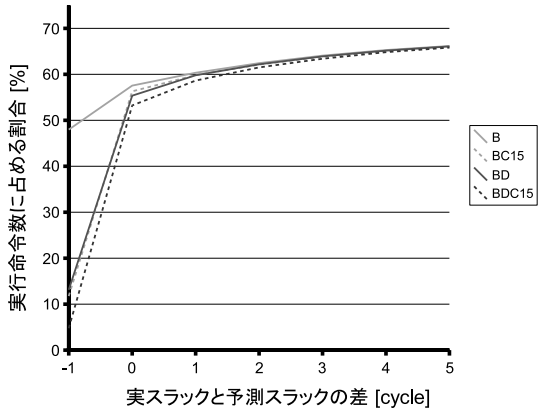

図 11 実スラックと予測スラックの差（$Vmax = 1$）
Fig. 11 Distance between real slack and predicted slack ($Vmax = 1$).

あれば，スラックを利用することができる．図より，予測ミスペナルティを発生させることなく，スラックを利用できる割合は，ヒット率が高いほど下がってしまうが，その変化は比較的緩やかであることが分かる．これらより，提案機構は，予測スラックが 1 以上となる割合を単純に減少させようとしているのではなく，主に予測ミスペナルティの発生率が減少するように，予測スラックを変化させていることが分かる．

次に，$Vmax$ の影響について考察する．図より，$Vmax$ を変化させた場合，予測スラックが 0 である割合，および，1 以上である割合はあまり変化しないことが分かる．このことから，スラックが 1 以上ある（またはスラックがない）と予測する命令の数は，$Vmax$ にあまり依存していないことが分かる．

また，予測スラックが 1 以上である命令の内訳は，$Vmax$ を 1 から 5 に増加させたときには変化するものの，$Vmax$ を 5 から 15 に増加させたときには，あまり変化しないことが分かる．これらより，$Vmax$ がある程度大きくなると，予測スラックと実スラックの大小関係はあまり変化しなくなることが分かる．

5.4 実スラックと予測スラックの差

前節の評価により，実スラックと予測スラックの大小関係を知ることができた．しかし，これだけでは，両者の差が実際にどの程度あるのかが分からない．

そこで，実スラックから予測スラックを引いた値の累積分布を測定する．測定では，まず，NO-DELAY モデルにおいて，各動的実行命令の実スラックをすべて採取する．そして，各提案モデルにおいて，採取した実スラックから，これに対応する予測スラックを引いた値を求める．

測定結果を，**図 11**，**図 12**，**図 13** に示す．これら

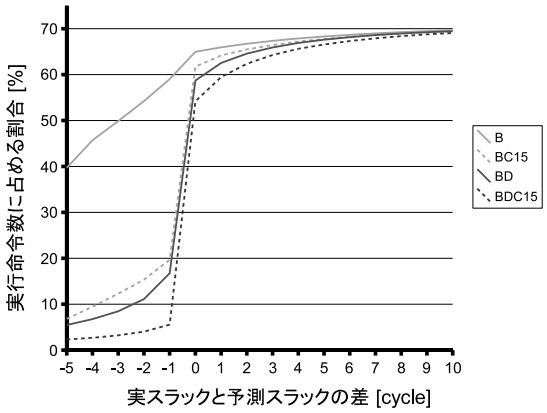

図 12 実スラックと予測スラックの差（$Vmax = 5$）
Fig. 12 Distance between real slack and predicted slack ($Vmax = 5$).

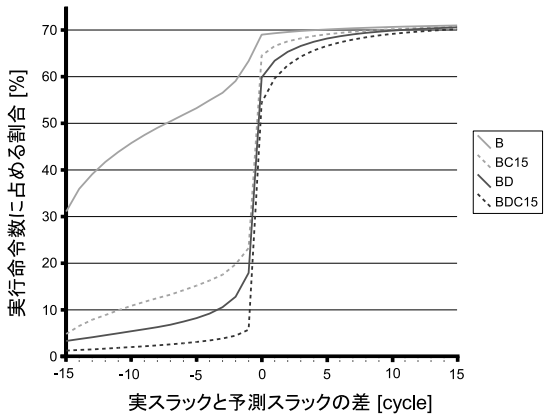

図 13 実スラックと予測スラックの差（$Vmax = 15$）
Fig. 13 Distance between real slack and predicted slack ($Vmax = 15$).

は，$Vmax$ を，それぞれ，1，5，15 とした場合である．図の縦軸は，全実行命令数に占める割合をベンチマーク平均で示し，横軸は実スラックから予測スラックを引いた値を示す．この値が負の場合，予測スラックが実スラックを上回っていることを示す．0 の場合，スラック予測がヒットしていることを示す．正の場合，予測スラックが実スラックを下回っていることを示す．横軸の最小値は，実スラックの最小値 0 から，$Vmax$ を引いた値となる．図 11 において，折れ線グラフは，一番上のグラフが B モデル，ほぼ重なっているグラフが BC15 モデルと BD モデル，一番下のグラフが BDC15 モデルの場合である．一方，図 12 と図 13 において，折れ線グラフは，上から順に，B モデル，BC15 モデル，BD モデル，BDC15 モデルの場合である．各モデルの比較を容易にするため，$Cth = 5$ の場合の結果は省略する．

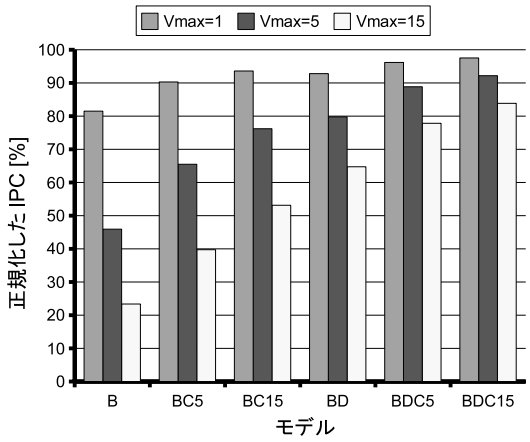

図 14 IPC
Fig. 14 IPC.

図 11〜図 13 より，予測スラックの減少や信頼性の導入を行うと，予測ミスペナルティの発生率だけでなく，予測ミスペナルティの大きさも抑制できることが分かる．

また，各モデルの差は，正の領域よりも，負の領域の方が大きい．これは，予測スラックを大きくする効果の差よりも，予測ミスペナルティを小さくする効果の差の方が大きいことを示している．このことから，予測スラックの減少の導入と，信頼性の導入は，目的どおりに，スラック予測ミスペナルティを削減できていることが分かる．

さらに，どのモデルにおいても，$Vmax$ が大きくなるほど，予測ミスペナルティが大きくなることが分かる．この原因は，実スラックが大幅に低下する命令が数多く存在することにある．たとえば，$Vmax = 15$ の場合，予測スラックの増減しか行わない B モデルにおいて，差が -15 になる命令は，31.1%となる．これは，実スラックが 15 以上減少した命令が，31.1%存在していることを示す．

5.5 性能に与える影響

図 14 に，各モデルの IPC を測定した結果を示す．縦軸は，NO-DELAY モデルの場合で正規化した IPC を，ベンチマーク平均で示す．横軸はモデルを示す．3 本で組になった棒グラフは，左から順に，$Vmax$ が 1，5，15 の場合である．

図 14 より，$Vmax$ が同一であるモデルどうしを比較すると，IPC は B モデルが最も低いことが分かる．また，予測スラックの減少や信頼性を単独で導入したモデルよりも，これらを組み合わせたモデル（BDCn モデル）の方が高い性能を達成することが分かる．なお，信頼性を導入したモデルの場合，信頼性の閾値（モ

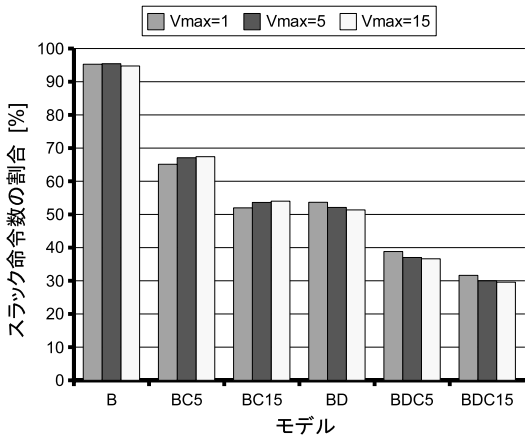

図 15 スラック命令の割合
Fig. 15 Slack instruction rate.

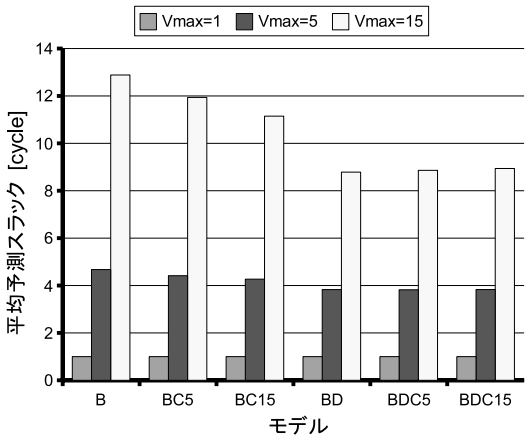

図 16 平均予測スラック
Fig. 16 Average predicted slack.

デルに付加した数字）が高いほど，性能は高くなる．

各モデルの性能が低下する原因は，スラック予測ミスペナルティの発生にある．そこで，上記の結果と，スラック予測精度を示した図 8〜図 10 を比べると，$Vmax$ が同一であれば，予測スラックが実スラックを上回る（予測ミスペナルティが発生する）割合が低いモデルほど，性能が高くなっていることが分かる．

図 14 より，各モデルにおいては，$Vmax$ を増加させるほど，IPC が低下することが分かる．しかし，IPC が高いモデルほど，IPC の低下率を抑制できることが分かる．この理由は，図 11〜図 13 から分かるように，予測スラックの減少や信頼性の導入を行うと，予測ミスペナルティの発生率だけでなく，予測ミスペナルティの大きさも抑制できるからである．

図 15 と**図 16** に，各モデルの予測スラックを評価した結果を示す．図 15 は，**スラック命令**の数を示す．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ある．縦軸は，全実行命令数に占めるスラック命令数の割合をベンチマーク平均で示し，横軸はモデルを示す．一方，図 16 は，**平均予測スラック**を示す．ここで，平均予測スラックとは，予測スラックの合計を，スラック命令数で割ることにより得られる値である．図の縦軸は，予測スラックの平均値をベンチマーク平均で示し，横軸はモデルを示す．これらの図より，実行レイテンシを増加させることのできた命令の割合と，それらの命令に対して，増加させることのできた実行レイテンシの平均を知ることができる．

図 15 より，スラック命令数は，モデルの種類や信頼性の閾値に依存し，IPC が高いモデルほど少なくなるが，$Vmax$ にほとんど依存しない．一方，図 16 より，平均予測スラックは，$Vmax$ が高いほど大きくなるが，モデルの種類や信頼性の閾値によって変化することはあまりないことが分かる．これらより，$Vmax$ が同一であるモデルどうしを比較すると，増加させた実行レイテンシの合計は，予測スラックの減少や信頼性を導入することで減少し，BDCn モデルにおいて最も少なくなる．また，信頼性を導入したモデルの場合，信頼性の閾値が高いほど，増加させた実行レイテンシの合計は少なくなる．

しかし，BDCn モデルは，$Vmax$ の増加による IPC の低下を最も抑制できる．そのため，性能をあまり低下させることなく，他のモデルよりも予測スラックを増やすことができる場合がある．たとえば，IPC の低下が 80%程度まで許されるような状況において，BC15 モデル，BD モデル，BDC15 モデルは，$Vmax$ を，それぞれ，5，5，15 まで増加させることができる．このとき，BDC15 モデルは，増加させることのできた実行レイテンシの合計が，BC15 モデルよりも 15.6%，BD モデルよりも 32.6%多くなる．

文献 10) では，従来手法によってローカル・スラックを予測し，それに基づいて命令の実行レイテンシを 1 サイクル増加させた場合の性能，および，スラック命令数を測定している．それによると，従来手法では，性能の低下が 2.8%のときに，スラック命令数の割合が 26.7%になる．

一方，上記の研究とはベンチマーク・プログラムやプロセッサ構成が異なるものの，本研究で最も近い評価を行っているのは，BDC15 モデルにおいて，$Vmax = 1$ とした場合である．この場合，性能の低下が 2.5%のときに，スラック命令数の割合が 31.6%となる．これより，提案手法は，従来手法と同様の結果を示すこと

が分かる．

6. スラック予測機構のハードウェアに関する評価

提案するスラック予測機構のハードウェア量，アクセス時間，消費電力を従来機構と比較する．

6.1 ハードウェア構成

プロセッサ構成は，前章の評価環境と同じものを用いる．また，提案機構として，前章で評価した BDC モデルを用いる．

まず，従来機構で必要となるハードウェアを以下に示す．

- テーブル
  - スラック表
  - メモリ定義表
  - レジスタ定義表
- 演算器
  - 減算器（スラック値の計算）
  - 比較器（アドレスの比較）
  - 比較器（物理レジスタ番号の比較）

従来機構において，スラック表は，命令のスラック値を保持しており，PC をインデクスとして，フェッチ時に参照し，実行時に更新する．メモリ定義表は，メモリ・アドレスをインデクスとし，対応するメモリ・アドレスにデータをストアした命令の PC と，そのデータの定義時刻を保持する．このテーブルは，ストア・アドレスで更新し，ロード・アドレスで参照する．レジスタ定義表は，物理レジスタ番号をインデクスとし，対応する物理レジスタにデータを書き込んだ命令の PC とそのデータの定義時刻を保持する．このテーブルは，命令の実行直前に，命令のソース・レジスタに対応する物理レジスタ番号で参照し，デスティネーション・レジスタに対応する物理レジスタ番号で更新する．減算器は，定義表から得られた定義時刻と現在時刻との差分をとり，実行した命令のスラックを計算する．比較器（アドレスの比較）と比較器（物理レジスタ番号の比較）は，それぞれ，メモリ定義表，レジスタ定義表を，高速動作のためにパイプライン化したときに必要となる．テーブルをパイプライン化した場合，定義時刻の更新が完了する前に，その定義時刻の参照が発生すると，テーブルから正しい定義時刻を得ることができない．この問題を解決するためには，定義時刻のフォワーディングを行う必要がある．具体的には，まず，更新に用いるアドレスと参照に用いるアドレスの比較，更新に用いるデスティネーション・レジスタと参照に用いるソース・レジスタの物理レジスタ番号の比較を行う．そして，アドレス，あるいは，物理レジスタ番号が一致すると，それぞれ，メモリ定義時刻，レジスタ定義時刻のフォワーディングを行う．

次に，提案機構で必要となるハードウェアを以下に示す．

- テーブル
  - スラック表
  - FIFO（信頼性と予測スラックの記録）
- 演算器
  - 加算器（信頼性）
  - 比較器（信頼性）
  - 加算器（予測スラック）
  - 比較器（予測スラック）

提案機構において，スラック表は，ある PC のスラック値と信頼性を保持しており，フェッチ時に参照し，コミット時に更新する．FIFO は，スラック表から得た信頼性と予測スラックを命令フェッチ順に保持する FIFO であり，ディスパッチ時に書き込み，コミット時に読み出す．これらの値は，スラック表の更新データを計算するために用いる．この FIFO は，ROB と同一エントリとし，命令を ROB に書き込むと同時に，同一のインデクスを用いて，その命令の信頼性と予測スラックをこの FIFO に書き込み，命令を ROB からコミットすると同時に，同一のインデクスを用いて，その命令の信頼性と予測スラックを FIFO から読み出す．

演算器は，更新のために用いる．加算器（信頼性）は，信頼性を $Cinc$ 増加させるために用いる．比較器（信頼性）は，増加させた信頼性が Cth 以上になったかどうかを調べるために用いる．加算器（予測スラック）は，予測スラックを $Vinc$ 増加させるために用いる．比較器（予測スラック）は，増加させた予測スラックが $Vmax$ を超えたかどうかを調べるために用いる．予測スラックが $Vmax$ を超えていたら，予測スラックは $Vmax$ にセットされる．なお，信頼性を減少させるときは，0 にリセットするだけなので，信頼性を減算するための演算器，および，信頼性が $Cmin$ 以下になったかどうかを調べるための比較器は必要ない．また，本評価では，$Vdec = Vmax$ としており，予測スラックを減少させるときは，0 にリセットするだけなので，予測スラックを減算するための演算器，および，予測スラックが $Vmin$ 以下になったかどうかを調べるための比較器も必要ない．

$Cinc$ と $Vinc$ は，ともに 1 なので，提案機構の加算器は，信頼性，あるいは，予測スラックだけを入力とし，その入力に 1 を加えるという非常に単純な操作

表 2 テーブルのコスト
Table 2 Cost of tables.

(a) 従来機構

	エントリ数	1 エントリあたりのメモリ・セル数		ポート数
		タグ・フィールド	データ・フィールド	
スラック表	E_{slack}	$32 - log_2(E_{slack}) + log_2(A_{slack})$	$log_2(Vmax + 1)$	$N_{fetch} + N_{issue}$
メモリ定義表	E_{mdef}	$32 - log_2(E_{mdef}) + log_2(A_{mdef})$	$32 + log_2(T_{cs})$	N_{dcport}
レジスタ定義表	E_{rdef}	$log_2(E_{preg}) - log_2(E_{rdef}) + log(A_{rdef})$	$32 + log_2(T_{cs})$	$3 \times N_{issue}$

(b) 提案機構

	エントリ数	1 エントリあたりのメモリ・セル数		ポート数
		タグ・フィールド	データ・フィールド	
スラック表	E_{slack}	$32 - log_2(E_{slack}) + log_2(A_{slack})$	$log_2(Vmax + 1) + log_2(Cth + 1)$	$N_{fetch} + N_{commit}$
FIFO	E_{rob}	-	$log_2(Vmax + 1) + log_2(Cth + 1)$	$N_{fetch} + N_{commit}$

を行うだけでよい．具体的には，入力の第 0 ビットから第 $n-1$ ビットがすべて 1 であれば，入力の第 n ビットを反転したものを，出力の第 n ビットとし，そうでなければ入力の第 n ビットをそのまま出力の第 n ビットとする．したがって，従来機構の減算器とは異なり，非常に簡単に実現できる．

$Cinc$ と $Vinc$ が，ともに 1 であることを利用すれば，提案機構の比較器も，簡単化できる．提案機構の加算器は，信頼性（or 予測スラック）に 1 を加えるだけである．したがって，比較器は，加算器の入力が，$Cth - 1$（or $Vmax$）と一致するのであれば，加算器の出力が Cth 以上になる（or $Vmax$ を超える）と判断することができる．

従来機構と提案機構を正確に比較するためには，それぞれの機構において，スラック予測精度がほとんど変化せず，アクセス時間と消費電力ができるだけ少なくなるようなテーブル構成（エントリ数，連想度，ラインサイズ，ポート数）を知る必要がある．しかし，従来機構において，テーブル（スラック表，メモリ定義表，レジスタ定義表）の構成がスラック予測精度に与える影響は，まだ十分に調査されていない．

そこで，本章では，従来機構と提案機構の精度が同程度になるテーブル構成を用いる．具体的には，スラック表に関しては，前章の評価で用いた構成（エントリ数 8K，連想度 2）を用いる．Cth と $Vmax$ は，どちらも，前章の評価において用いた値において，提案機構のハードウェア量が最も大きくなる値である，15 を仮定する．メモリ定義表とレジスタ定義表に関しては，前章で精度を比較するために引用した文献 10) で仮定されている構成を用いる．具体的には，メモリ定義表はエントリ数が 8 K，連想度が 4，レジスタ定義表はエントリ数が 64，連想度が 64 とする．

文献 10) によれば，定義表は PC の一部を保持する．また，前章の評価結果からも分かるように，動的実行命令のうち，実スラックが 30 以下である命令は約 7 割存在するため，定義時刻を表すために必要なビット数を少なくできる可能性がある．しかし，文献 10) において，これらの数値に関する具体的な議論は行われていない．そこで，本章では，スラック予測精度を重視し，定義表には PC をすべて保持すると仮定する．また，定義時刻を表すために必要なビット数の削減は行わないと仮定する．したがって，定義表の各データ・フィールドは，最悪のケースを想定した設定となる．

上記のテーブル構成は，スラック予測精度を重視した構成であるため，アクセス時間と消費電力が過大になる可能性がある．しかし，精度がほぼ同一になることが判明しているテーブル構成を用いて，アクセス時間と消費電力を比較できるという利点がある．

6.2 ハードウェア量の比較

ハードウェア量の比較は，必要となるテーブルの保持するメモリ・セル数，および，演算器の入力ビット数と個数を基にして行う．

テーブルにおいて，ハードウェア量の大部分を占めるのは，タグ・アレイとデータ・アレイである．そこで，テーブルのハードウェア量を，タグ・アレイとデータ・アレイの保持するメモリ・セル数によって見積もる．

表 2 に，必要となるテーブルのメモリ・セル数とポート数を示す．表 2 (a) は従来機構の場合，(b) は提案機構の場合である．表には，まず，各テーブルのエントリ数を示し，次に，1 エントリあたりのメモリ・セル数を，タグ・フィールドとデータ・フィールドに分けて示す．エントリ数と，1 エントリあたりのメモリ・セル数の積が，テーブルの総メモリ・セル数となる．また，表には，各テーブルのポート数も示す．ポート数は，後ほどアクセス時間と消費電力について評価するために用いる．表では，スラック表，メモリ定義表，レジスタ定義表のエントリ数を，それぞれ，E_{slack}，E_{mdef}，E_{rdef} と表し，連想度を，それぞれ，A_{slack}，A_{mdef}，A_{rdef}

と表す．同じ条件で比較するため，スラック表のエントリ数と連想度は，提案機構と従来機構で同一としている．N_{fetch}，N_{issue}，N_{dcport}，N_{commit} は，それぞれ，フェッチ幅，発行幅，データ・キャッシュのポート数，コミット幅を表す．N_{fetch}，N_{issue}，N_{commit} は同一と仮定する．E_{rob} は，ROB のエントリ数を表す．前章の評価環境より，$N_{fetch} = 8$，$E_{ROB} = 256$ とする．

T_{cs} は，コンテキスト・スイッチ間隔をサイクル単位で表した値である．従来機構では，時刻を用いてスラックを計算する．スケジューラによって選択されたプロセスが実行を開始した時刻を 0 とすると，その時刻は，プロセスがコンテキスト・スイッチによってプロセッサ上から退避されるまでカウントされる．したがって，時刻を正しく表すためには，$log_2(T_{cs})$ ビット必要となる．Linux OS において，コンテキスト・スイッチの間隔は ms オーダなので，T_{cs} を 1 ms 程度と仮定する．また，文献 9) に示された，0.13 μm プロセス時の ARM コアの動作周波数より，プロセッサの動作周波数を 1.2 GHz と仮定する．これらより，時刻を表現するにはほぼ 20 ビット必要となる．そこで，以降は，$log_2(T_{cs}) = 20$ とする．

従来機構と提案機構のスラック表を比較すると，従来機構は，データ・フィールドのメモリ・セル数が $log_2(Cth + 1)$ ビット多くなる．しかし，スラック表以外にもテーブルは存在するので，スラック表だけでは全テーブルのハードウェア量の大小を判断できない．

そこで，表の各変数に値を代入して全テーブルのハードウェア量を計算する．提案機構のメモリ・セル数は，スラック表の場合 229,376，FIFO の場合 2,048 となり，合計すると 231,424 となる．一方，従来機構のメモリ・セル数は，スラック表の場合 196,608，メモリ定義表の場合 598,016，レジスタ定義表の場合 3,840 となり，合計すると 798,464 となる．したがって，提案機構の方が，メモリ・セル数が少なくなる．

なお，上記の評価において，従来機構の定義表は，各データ・フィールドのサイズが，最悪のケースを想定した設定となっているが，このサイズが半分になったとしても，提案機構のメモリ・セル数の方が少ないという結論は変わらない．ただし，前節で説明したとおり，正確な比較を行うためには，十分なスラック予測精度が得られるテーブル構成を知る必要があり，今後の課題である．

次に，演算器のハードウェア量を比較する．**表 3** に，演算器の入力ビット数と個数を示す．表 3 (a) は従来機構の場合，(b) は提案機構の場合である．入力ビット数は，演算器の各入力のビット数を合計したものである．なお，比較器の個数は，定義時刻のフォワーディングを行うパイプライン段数が 1 段の場合の値である．段数が増加すれば，それに比例して，比較器の個数も増加するが，フォワーディングを行う必要がなければ，比較器も必要ない．

表 3 演算器のコスト
Table 3 Cost of operation units.

(a) 従来機構

	個数	入力ビット数
減算器	N_{issue}	$log_2(T_{cs})$
比較器（アドレス）	$(N_{dcport})^2$	32
比較器（レジスタ番号）	$(N_{dcport})^2$	$log_2(E_{preg})$

(b) 提案機構

	個数	入力ビット数
加算器（信頼性）	N_{commit}	$log_2(Cth + 1)$
比較器（信頼性）	N_{commit}	$log_2(Cth + 1)$
加算器（予測スラック）	N_{commit}	$log_2(Vmax + 1)$
比較器（予測スラック）	N_{commit}	$log_2(Vmax + 1)$

従来機構と提案機構の演算器を比較する．ここでは，提案機構の方が確実にハードウェア量が少なくなることを示すため，従来機構において，定義時刻のフォワーディングは行う必要がない場合を考える．

$N_{issue} = N_{commit} = 8$ なので，提案機構は，従来機構よりも，演算器の数が 24 個，多くなってしまうことが分かる．しかし，提案機構の演算器は，先ほど説明したとおり，非常に簡単に実現できるため，単純に演算器の個数だけに着目して，ハードウェア量を比較することはできない．

そこで，各演算器の構成について詳しく検討する．まず，従来機構の減算器は，$log_2(T_{cs}) = 20$ なので，入力が 20 ビットとなる．基本的な回路構成は，入力が 20 ビットの加算器とほぼ同様である．この加算器を 8 倍したものが，従来機構のハードウェア量となる．

次に，提案機構の演算器の構成について詳しく検討する．まず，Cth と $Vmax$ を，先ほどと同様，どちらも 15 と仮定すると，提案機構の各演算器は，入力が 4 ビットとなる．この場合の各演算器の構成を，**図 17** に示す．図は，コミットする命令 1 つあたりに必要となる演算器（これらで構成される回路を更新ユニットと呼ぶこととする）の回路構成を示す．この回路を 8 倍したものが，提案機構のハードウェア量となる．図の R flag は，ターゲット・スラック到達条件が成立しているときに 1，そうでないときに 0 となるフラグである．図の左中央の AND ゲートが比較器，点線で囲われた部分が加算器，その他が制御用の回路である．入力ビット数が 4 ビットの場合，比較器は，入力値の

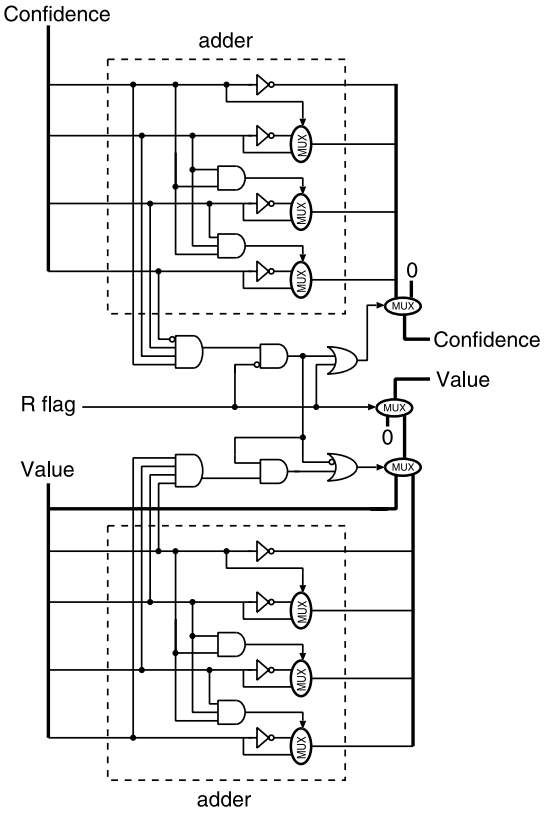

図 17 更新ユニット
Fig. 17 Update unit.

各ビットを，そのまま，あるいは，反転したものを入力とする，4 入力 AND ゲートで実現できる．また，提案機構の加算器は，2 個の AND ゲート，4 個のインバータ，3 つの MUX で実現できる．したがって，従来機構で必要となる 20 ビットの減算器よりも十分少ないハードウェア量で実現できるといえる．

6.3 アクセス時間と消費電力に関する比較

本節では，テーブルのアクセス時間と，1 アクセスあたりの消費エネルギーを求めるために，キャッシュ・シミュレータである CACTI[12] を用いる．CACTI による評価では，文献 9) の ARM コアのデータを基に，プロセスを 0.13 μm，電源電圧を 1.1 V と仮定する．CACTI ではテーブルのラインサイズをバイト単位で入力する必要がある．しかし，従来機構のスラック表は，データ・フィールドが 4 ビットであるため，ラインサイズが 1 バイトに満たない．そこで，CACTI で評価する場合に限り，データ・フィールドを 8 ビットと仮定する．しかし，この仮定によって，従来機構のスラック表の規模だけが 2 倍になってしまうため，このままでは公平な比較ができない．そこで，CACTI で提案機構を評価する場合，スラック値を保持するテーブルである，スラック表と FIFO のデータ・フィールドを，8 ビットから，16 ビットに増加させる．なお，メモリ定義表とレジスタ定義表は，スラック値を保持しないので，データ・フィールドの変更は行わない．

表 4 テーブルのアクセス時間
Table 4 Access time of tables.

(a) 従来機構

	アクセス時間
スラック表	4.85 ns
メモリ定義表	1.94 ns
レジスタ定義表	1.67 ns

(b) 提案機構

	アクセス時間
スラック表	5.05 ns
FIFO	0.50 ns

上記の仮定によって，提案機構のスラック表は，アクセス時間が 4.1%，消費エネルギーが 23%増加する．このことから，従来機構のスラック表の評価結果も，同程度の誤差が生じていると考えられる．また，提案機構の FIFO はアクセス時間が 4.2%減少し，消費エネルギーが 116%増加する．そこで，比較を行う際には，この誤差の影響を考慮する．なお，FIFO のアクセス時間が減少する理由は，CACTI が，テーブル構成によって，データ・アレイの分割方法を変えることにある．

まず，提案機構と従来機構のアクセス時間を比較する．すでに示したように，スラック予測機構で使用される演算器の規模は，ALU よりも小さい．一方，テーブルに関しては，プロセッサ中で用いられるデータ・キャッシュと同程度（あるいはそれ以上）の規模のものが存在する．そのため，提案機構と従来機構のアクセス時間は，テーブルのアクセス時間で決まると考えることができる．そこで，テーブルのアクセス時間を比較する．

表 4 に，CACTI で測定した，テーブルのアクセス時間を示す．表 4 (a) が従来機構の場合，(b) は提案機構の場合である．表より，スラック表は，メモリ定義表に比べ，ハードウェア量が少ないにもかかわらず，アクセス時間が非常に長くなっていることが分かる．この理由は，テーブルのアクセス時間が，ハードウェア量ではなく，テーブル構成（エントリ数，連想度，ラインサイズ，ポート数，など）で決まることにある．

また，動作周波数は 1.2 GHz（サイクル時間 0.83 ns）を仮定しているので，スラック表，メモリ定義表，レジスタ定義表を高速にアクセスするためには，それぞれを，6 段，3 段，2 段程度にパイプライン化する必要

があることが分かる．スラック表のアクセス時間の測定誤差を考慮したとしても，この段数が減ることはない．しかし，テーブルを 6 段にパイプライン化したとしても，フェッチした命令の予測スラックを得るために要するサイクル数が非常に長く，それを利用することは困難である．また，メモリ定義表，レジスタ定義表をパイプライン化すると，定義時刻のフォワーディングを行うために，消費電力が増加してしまうという問題がある．しかし本節では，テーブルを上記のようにパイプライン化したとして議論をすすめ，これらの問題については，次節で議論する．

さらに，表 4 より，どちらの機構においてもスラック表のアクセス時間が最も長いことが分かる．したがって，アクセス時間は，提案機構の方が長いことが分かる．スラック表のアクセス時間には測定誤差があるが，どちらの機構においてもアクセス時間は同程度増加すると考えられるので，この結論に影響はない．

次に，消費電力の比較を行う．ただし，前章の評価結果より，従来機構と提案機構の実行時間はほぼ同一となるので，消費エネルギーを比較すればよい．回路の全消費エネルギーは，1 回の動作あたりに必要な消費エネルギーと，動作回数の積で表される．

各回路の動作回数は，前章の評価環境を用いて計測する．前章で用いたシミュレータに従来機構は組み込まれていないので，従来機構の各回路の動作回数は，プロセッサの動作から推測する．具体的には，スラック表の場合，フェッチ時に参照し，命令実行時に更新するので，フェッチした命令数と機能ユニットで実行した命令数の和を動作回数とする．メモリ定義表の場合，ロード命令の実行時に参照し，ストア命令の実行時に更新するので，ロード/ストア命令の実行回数を，動作回数とする．レジスタ定義表の場合，実行する命令のソース・レジスタに対応する物理レジスタ番号で参照し，デスティネーション・レジスタに対応する物理レジスタ番号で更新するので，機能ユニットで実行した命令のソース・レジスタ数とデスティネーション・レジスタ数の和を，動作回数とする．減算器の場合，時刻からスラックを計算する可能性のある命令，つまり，機能ユニットで実行した，デスティネーション・レジスタを持つ命令とストア命令の数の和を，動作回数とする．従来機構の比較器については，パイプライン化されたメモリ定義表，および，レジスタ定義表が存在すると仮定して，各サイクルにおいて，どの命令がどのテーブルの参照/更新を行っているのかをシミュレーションする．そして，同じテーブルに対して参照/更新を行う命令間で，定義時刻のフォワーディングのために必要となる，メモリ・アドレスの比較，あるいは，物理レジスタ番号の比較を行い，その比較回数を，それぞれ，アドレス用比較器，レジスタ番号用比較器の動作回数とする．サイクル時間は 0.83 ns を仮定しているので，表 4 より，メモリ定義表とレジスタ定義表は，それぞれ，3 段，2 段にパイプライン化すると仮定する．

表 5 消費エネルギー
Table 5 Energy consumption.

(a) 従来機構

	動作回数	1 動作あたりの消費エネルギー
スラック表	322 M	4.33 nJ
メモリ定義表	52 M	1.33 nJ
レジスタ定義表	261 M	1.12 nJ
減算器	111 M	-
比較器（アドレス）	27 M	-
比較器（レジスタ番号）	488 M	-

(b) 提案機構

	動作回数	1 動作あたりの消費エネルギー
スラック表	288 M	5.37 nJ
FIFO	278 M	0.28 nJ
更新ユニット	100 M	-

1 動作あたりの消費エネルギーは，テーブルの場合 CACTI を用いて測定する．一方，演算器の場合，前節で示したハードウェア量を基に，どちらの消費エネルギーの方が大きくなるのかを検討する．

表 5 に，各回路の動作回数のベンチマーク平均と，テーブルの 1 動作あたりの消費エネルギーを示す．表 5 (a) は従来機構の場合，(b) は提案機構の場合である．

まず，演算器の消費エネルギーを比較する．ここで，演算器の 1 動作あたりの消費エネルギーは，1 回の動作で充放電する負荷容量の平均と，電源電圧の 2 乗の積で表される．電源電圧は一定である．一方，充放電する負荷容量は，動作時にスイッチしたノードの全容量で表される．この値を正確に求めるためには，演算器を設計し，与えられた入力に対してどのノードがスイッチしたかを調べる必要があり，容易に評価することができない．そこで本節では，比較を簡単に行うために，ハードウェア量が多いほど充放電する負荷容量も増えると仮定する．そして，前節で示したハードウェア量を基に，演算器の 1 動作あたりの消費エネルギーを比較する．

前節より，提案機構の演算器（更新ユニット）のハードウェア量は，従来機構の減算器よりも十分少ない．そのため，提案機構の演算器を 1 回動作させるために

必要な消費エネルギーも少ないと判定することができる．また，表 5 より，演算器の動作回数は提案機構の方が少ない．これらより，提案機構の演算器の全消費エネルギーは，従来機構の減算器よりも少なくなると考えられる．

さらに，従来機構では，定義時刻のフォワーディングを行う必要がある．具体的には，配線を用いて，比較値（アドレス，あるいは，レジスタ番号）と定義時刻をブロードキャストし，比較器を用いて，アドレス比較，あるいは，レジスタ番号比較を行い，比較結果が一致すれば，MUX を介して，対応する定義時刻を減算器に供給するという操作を行う．そのため，1 動作あたりの消費エネルギーは無視できないレベルになると考えられる．また，表 5 より，アドレスの比較回数とレジスタ番号の比較回数は，それぞれ，27 M，488 M と多い．

これらより，提案機構の演算器の全消費エネルギーは，従来機構の演算器（減算器，比較器，および，ブロードキャスト用の配線）の全消費エネルギーよりも，かなり少なくなると考えることができる．

次に，テーブルの消費エネルギーを比較する．役割がほぼ同じであるスラック表において，1 動作あたりの消費エネルギーは，従来機構の方が少なく，動作回数は，提案機構の方が少ないが，スラック表の全消費エネルギーは，従来機構の方が少なくなる．しかし，テーブル全体の消費エネルギーを合計すると，従来機構の場合 1.76J，提案機構の場合 1.62J となり，提案機構の方が少なくなることが分かる．

ここで，CACTI の測定誤差の影響について考える．スラック表の消費エネルギーには測定誤差があるが，どちらの機構においても消費エネルギーは同程度増加すると考えられるので，スラック表の比較結果に影響はないといえる．また，測定誤差によって FIFO の消費エネルギーはより大きく見積もられるが，メモリ定義表とレジスタ定義表の消費エネルギーに測定誤差は生じない．これらより，テーブル全体の消費エネルギーに与える影響を考えると，発生する測定誤差は，提案機構に対してより不利に働く．したがって，提案機構の消費エネルギーの方が小さいという結論は変わらないといえる．

以上より，消費エネルギーは，演算器とテーブル，いずれにおいても，従来機構の方が大きくなると考えられる．

従来機構のスラック表は，提案機構よりも消費エネルギーが小さい．そのため，スラック予測精度を低下させることなく，メモリ定義表とレジスタ定義表の消費エネルギーを削減することができれば，全テーブルの消費エネルギーを提案機構よりも小さくできる可能性がある．この目的を達成するアプローチとして，回路中で用いられるトランジスタのサイズを小さくし，充放電する負荷容量を減らすという方法が考えられる．この方法では，テーブル構成を変えなくてもよいため，スラック予測精度を低下させることなく，消費エネルギーを削減できる．

しかしこのアプローチでは，トランジスタのサイズを小さくするので，メモリ定義表とレジスタ定義表のアクセス時間が長くなってしまう．その結果，これらのテーブルは，パイプライン段数が増加し，定義時刻のフォワーディングに要する消費エネルギーが増加してしまう．このように，高速アクセスに必要となる，定義時刻のフォワーディングは，演算器の消費エネルギーを増大させるだけでなく，上記アプローチによる消費エネルギーの削減も妨げていることが分かる．

6.4 参照の局所性を利用したテーブル構成の最適化

前節で用いたテーブル構成は，アクセス時間が非常に長くなるため，予測スラックの利用を困難にするという問題と，定義時刻のフォワーディングに関する消費エネルギーを増加させるという問題を引き起こす．これらの問題を解決するためには，テーブル構成（エントリ数，連想度，ラインサイズ，ポート数）を変更する必要がある．しかし，6.1 節で述べたとおり，従来機構において，テーブル構成がスラック予測精度に与える影響は明らかにされていない．そのため，テーブル構成を単純に変化させて，アクセス時間や消費電力を測定することはあまり意味がない．

そこで本節では，前節で用いたテーブル構成に対して，スラック予測精度に与える影響が少ないと考えられる変更のみを行い，アクセス時間と消費電力がどのように改善されるのかを評価する．なお，提案機構の FIFO は，他のテーブルよりもアクセス時間が十分に短いため，構成は変更しない．

この目的のために，我々は，各テーブルのアクセス・パターンに着目する．まず，スラック表について，データの参照時と更新時のパターンに分けて考える．スラック表の参照では，フェッチする命令の PC をインデクスとして用いる．そのため，命令キャッシュと同様，インデクスとして用いる PC は，taken と予測した分岐に到達するまで連続しており，参照の局所性が非常に高い．

一方，スラック表の更新では，従来機構の場合，機能ユニットで実行した命令の PC をインデクスとして用いる．したがって，インデクスとして用いる PC は，

アウト・オブ・オーダ実行によって不連続になるものの，順番が入れ替わる範囲はプロセッサ内の命令に限られるので，依然として参照の局所性は高いといえる．また，提案機構の場合，ROB からコミットした命令の PC をインデクスとして用いる．したがって，インデクスとして用いる PC は，taken 分岐に到達するまで連続しており，更新の局所性は非常に高い．

以上より，スラック表においては，スラック予測精度にほとんど影響を与えることなく，ラインサイズを増やすことができると考えられる．ただし，キャッシュと同様，ラインサイズを増やしすぎると，ラインの利用効率が下がり，テーブルのミス率が増加するので，そのことを考慮して，ラインサイズを決定する必要がある．

さらに，インデクスとして用いる PC が連続していることを利用し，ライン単位で参照/更新を行うようにすれば，リード・ポートとライト・ポートの削減ができると考えられる．

ここで，スラック表のラインサイズを増やし，1 ライン上に 2 命令分のスラック値を保持した場合，ライン単位で参照/更新を行うと，リード・ポートとライト・ポートがどれだけ削減できるかを考える．本節で想定しているプロセッサでは，$N_{fetch} = 8$ なので，ライン単位で参照/更新を行うのであれば，ポート数は 10 本（リード・ポート 5 本，ライト・ポート 5 本）まで削減できる．それ以上ポートが存在しても，使用することはできない．また，参照/更新の対象となるスラック値は，必ずしもラインの先頭から順番に並んでいるわけではないので，ポート数をさらに 8 本まで減らすと，参照/更新に失敗する場合が発生してしまう．これらより，ラインサイズが決まれば，削減できるポート数は一意に決まることが分かる．

同様にして，さらにラインサイズを増加させた場合について考えると，1 ライン上に 4 命令分，8 命令分のスラック値を保持した場合，ポート数は，それぞれ，6 本，4 本となることが分かる．ただし，それを超えてラインサイズを増やしても，参照/更新の対象となるスラック値は 2 つのラインに分かれて存在する可能性があるため，ポート数を 4 本よりも小さくすることはできない．なお，従来機構の場合，更新時にインデクスとして用いる PC は連続していないので，ライン単位で更新を行ってもライト・ポートの削減はできない．しかし，更新データをバッファに蓄え，そこからフェッチ順に更新していくという変更を加えれば，比較的容易に更新データの整列ができると考えられる．そこで本節では，従来機構においても，ライト・ポートの削減は可能であると仮定する．

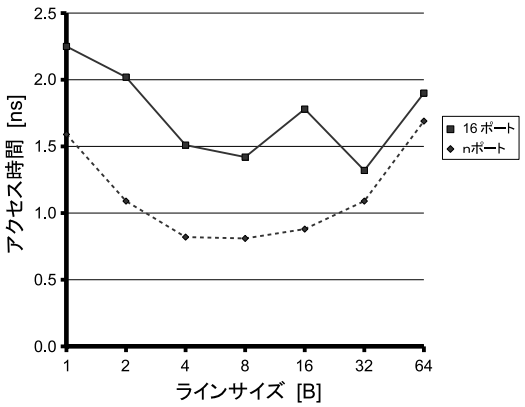

図 18 スラック表のアクセス時間（従来機構）
Fig. 18 Access time of slack table (conventional scheme).

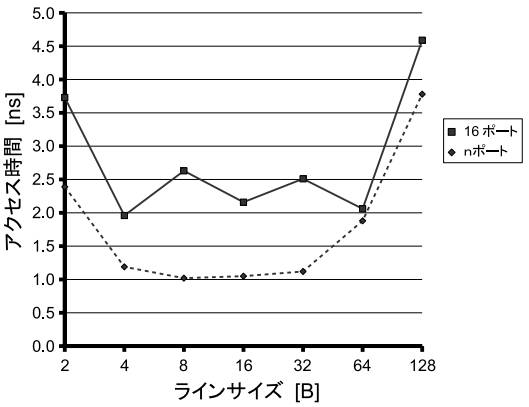

図 19 スラック表のアクセス時間（提案機構）
Fig. 19 Access time of slack table (our proposal).

図 18 と**図 19** に，それぞれ，従来機構と提案機構において，スラック表のラインサイズを，2^n 倍（$1 \leq n \leq 7$）に増加させて，アクセス時間を評価した結果を示す．評価には CACTI を用いる．前節で説明したように，従来機構のスラック表は，データ・フィールドが 4 ビットであるため，ラインサイズを増加させない場合，CACTI で評価できない．しかし，上記のようにラインサイズを増加させれば，ラインサイズはバイト単位で増加していくため，CACTI で評価できるようになる．そこで，本節では，前節とは異なり，データ・フィールドのビット数を変更せずに評価を行う．これにより，前節よりも正確に従来機構と提案機構の比較ができるようになる．

図の縦軸は，アクセス時間を示し，横軸はラインサイズを示す．折れ線グラフは，上側がポート数を削減しない場合，下側がポート数を削減する場合である．図から分かるように，ポート数を減らすとアクセス時

間が短くなる．一方，アクセス時間は，ラインサイズの増加にともない最初は減少するが，しばらくすると，逆に増加していくという傾向にあることが分かる．したがって，アクセス時間を減らすのであれば，ポート数を減らし，1 ライン上に 8 命令分か 16 命令分のスラック値を保持すればよいことが分かる．しかし，1 ライン上に，16 命令分以上のスラック値を保持しても，これらがすべて同時に必要になることはなく，ラインの利用効率が下がる．そこで，本節では，スラック表のラインサイズを，8 命令分のスラック値を保持できるサイズに変更し，ポート数を削減する．具体的には，従来機構の場合 4 B，提案機構の場合 8 B とする．このとき，いずれの機構においても，ポート数は 4 本に削減できる．

次に，メモリ定義表について考える．メモリ定義表は，ロード・アドレスとストア・アドレスをインデクスとして，それぞれ，参照，更新を行う．したがって，データ・キャッシュと同様，参照の局所性が高いといえる．そのため，スラック予測精度にほとんど影響を与えることなく，ラインサイズを増やすことができると考えられる．ただし，先ほどと同様，ラインサイズを増やしすぎないようにする必要がある．

図 20 に，メモリ定義表のラインサイズを変化させて，アクセス時間を評価した結果を示す．縦軸は，アクセス時間を示し，横軸はラインサイズを示す．図から分かるように，アクセス時間は，ラインサイズの増加にともない減少するが，28 B の時点で減少しなくなり，112 B 以上では増加してしまうことが分かる．したがって，アクセス時間を減らすのであれば，56 B を超えないようにラインサイズを増やせばよい．

しかし，ラインサイズを増やしすぎると，ラインの利用効率が下がり，テーブルのミス率が増加してしまう可能性がある．文献 7) によれば，容量が 1 K〜256 KB のデータ・キャッシュにおいて，ラインサイズを 16 B から 256 B まで増加させていった場合，32 B までであれば，どの容量においても，キャッシュ・ミス率が低下していくことが示されている．この場合，最小のブロックは 4 B なので，ラインサイズが 32 B の場合，1 ライン上に 8 ブロック分のデータを保持することを意味する．ベンチマーク等の評価環境が異なるが，本節では，この結果を参考に，テーブルのミス率を増加させないラインサイズの範囲を仮定する．具体的には，メモリ定義表では，最小のブロックは 7 B（PC ＋定義時刻）なので，56 B 以下のラインサイズであれば，テーブルのミス率は増加しないと仮定する．以上より，本節では，メモリ定義表のラインサイズを 56 B に変更する．

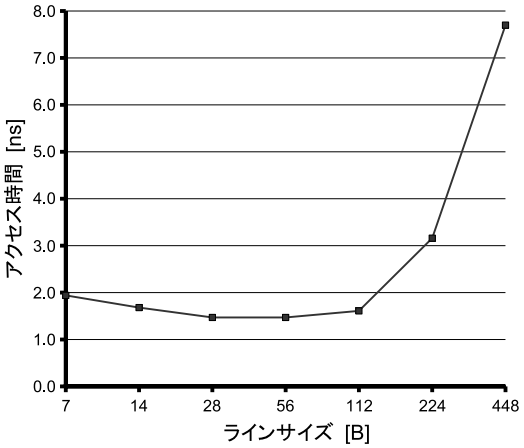

図 20 メモリ定義表のアクセス時間
Fig. 20 Access time of memory definition table.

表 6 テーブル構成変更後のアクセス時間と消費エネルギー
Table 6 Accesss time and energy consumption in modified table configurations.

(a) 従来機構

	アクセス時間	1 動作あたりの消費エネルギー
スラック表（4 B-line，4-port）	0.82 ns	0.22 nJ
メモリ定義表（56 B-line）	1.47 ns	1.09 nJ

(b) 提案機構

	アクセス時間	1 動作あたりの消費エネルギー
スラック表（8 B-line，4-port）	1.02ns	0.32nJ
FIFO（1 B-line，16-port）	0.52 ns	0.13 nJ

最後に，レジスタ定義表について考える．レジスタ定義表は，命令に割り当てられた物理レジスタ番号をインデクスとして，命令の実行直前に参照，更新する．したがって，スラック表や，メモリ定義表のような参照の局所性はない．そのため，本節では，レジスタ定義表の構成は変更しない．

表 6 に，参照の局所性に着目して，テーブル構成を最適化した場合の，アクセス時間と 1 動作あたりの消費エネルギーを示す．なお，本節では，CACTI で評価する際に，前節で行ったようなデータ・フィールドのビット数の変更を行う必要がない．そこで，そのような変更を行わなかった場合の FIFO についても，アクセス時間と 1 動作あたりの消費エネルギーを示す．

表より，従来機構と提案機構のスラック表は，どち

らも，アクセス時間が大幅に減少し，仮定しているサイクル時間 0.83 ns と非常に近い値となることが分かる．これにより，パイプライン段数は，従来機構の場合 1 段に，提案機構の場合 2 段に減少するので，フェッチした命令のスラック値を利用することは十分可能になる．また，メモリ定義表のアクセス時間が減少し，パイプライン段数が 3 段から 2 段になる．これにより，アドレス用比較器の数が段数分減少し，この比較器を動作させる回数が 27 M から 13 M に減少する．しかし，依然として定義時刻のフォワーディングは必要であり，演算器の全消費エネルギーは，従来機構の方が大きくなる．

また，表 6 より，スラック表とメモリ定義表のどちらにおいても，1 動作あたりの消費エネルギーが削減されていることが分かる．

次に，スラック予測機構全体のアクセス時間と消費エネルギーについて考える．表 4 と表 6 より，スラック表のアクセス時間が減少したことによって，従来機構のアクセス時間の方が，提案機構よりも長くなったことが分かる．

表 5 と表 6 を用いて，テーブル構成を最適化した後の消費エネルギーを計算する．なお，スラック表には，ライン単位で参照/更新を行い，ポート数を 4 分の 1 にするという変更が加えられているので，スラック表の動作回数は，表 5 に示された値の 4 分の 1 になると仮定して計算を行う．計算の結果，テーブル全体の消費エネルギーは，従来機構の場合 0.37J，提案機構の場合 0.06J となり，どちらも大幅に減少していることが分かる．なお，前節と同様，スラック表の消費エネルギーは従来機構の方が小さくなり，テーブル全体の消費エネルギーは提案機構の方が小さくなる．

以上より，参照の局所性を利用して，テーブル構成を最適化することで，スラック表のアクセス時間に関する問題は解決できることが分かる．また，スラック予測機構の消費エネルギーを大幅に削減できることが分かる．

7. 機能ユニットの消費電力の削減

ローカル・スラック予測の応用例として，予測スラックが 1 以上ある命令を，より低速でより消費電力の低い機能ユニットで実行することで，性能を大きく低下させることなく機能ユニットの低消費電力化をはかる研究が行われている[6]．そこで，本論文においても，上記の低消費電力化を応用例として取り上げ，提案手法の効果を評価する．

7.1 評価環境

5 章で述べた評価環境との違いについて述べる．

整数用の機能ユニット（iALU）として，高速なものと低速なものを 2 種類用意する．文献 9) によると，0.13 μm プロセスにおける ARM コアは，動作周波数が 1.2 GHz，600 Mhz の場合，電源電圧が，それぞれ，1.1 V，0.7 V であることが示されている．これを基に，プロセッサの動作周波数を 1.2 GHz（サイクル時間 0.83 ns）とし，高速な iALU と低速な iALU は，実行レイテンシがそれぞれ 1 サイクル，2 サイクル，電源電圧がそれぞれ 1.1 V，0.7 V とする．評価では，高速な iALU を n 個持つモデルを，$(n\mathrm{f}, (6-n)\mathrm{s})$ モデルと呼ぶこととする．

提案手法を用いてローカル・スラックを予測する．従来手法に近い条件で評価を行うため，$Vmax = 1$，$Cth = 15$ とし，スラック表のパラメータをすべて固定する．命令スケジューラが，オペランドの揃った命令から，iALU で実行する命令を選択した後，選択した命令において，予測スラックが 1 である命令を低速な iALU に，予測スラックが 0 である命令を高速な iALU に割り当てる．ただし，低速な iALU に空きがなければ高速な iALU に，高速な iALU に空きがなければ低速な iALU に命令を割り当てる．予測スラックは，命令を iALU に割り当てるときにのみ利用し，その他の処理では利用しない．たとえば，命令スケジューラが，iALU で実行する命令を選択する際に，予測スラックを利用することはない．また，命令を iALU に割り当てる順番は，命令スケジューラが命令を選択した順番に従っており，予測スラックを利用することはない．

上記の手法では，低速な iALU で命令を実行することで，iALU の消費エネルギーを削減する．しかし，予測スラックが実スラックを上回っている場合は，プロセッサ性能に悪影響を与えてしまう．プロセッサにおいて，性能は非常に重要な要素である．そこで，消費エネルギー削減効果と，プロセッサ性能への悪影響を同時に考慮できる指標として，消費エネルギーと，プロセッサの実行時間の積（EDP：Energy Delay Product）を測定する．

プロセッサの実行時間は，実行サイクル数とサイクル時間（動作周波数の逆数）の積で表すことができる．一方，機能ユニットの消費エネルギーは，iALU で命令を実行した回数と，1 実行あたりの消費エネルギーの積で表すことができる．1 実行あたりの消費エネルギーは，1 回の実行で充放電する負荷容量の平均と，電源電圧の 2 乗の積で表すことができる．したがって，

EDP は，以下の式で表される：

$$EDP = (C_f \cdot V_f^2 \cdot N_f + C_s \cdot V_s^2 \cdot N_s) \cdot N_c / f$$

ここで，C_f，C_s は，それぞれ，高速な iALU，低速な iALU において，1 実行あたりに充放電される負荷容量である．V_f，V_s は，それぞれ，高速な iALU，低速な iALU の電源電圧である．N_f，N_s は，それぞれ，高速な iALU，低速な iALU において命令を実行した回数であり，N_c は実行サイクル数である．f は動作周波数である．

V_f，V_s，f は，先程仮定した値を用いる．N_f，N_s，N_c は，シミュレーションによって求める．高速な iALU と低速な iALU は，動作周波数と電源電圧が異なるが，実行できる命令の種類は同じである．そこで本節では，ある動的命令をどちらの iALU で実行したとしても，その命令の実行を完了するまでに充放電する負荷容量（動作時にスイッチするノードの全容量）は同じであると仮定し，$C_f = C_s$ とする．

なお，厳密には，回路中でスイッチするノードは，演算の種類（加算，シフト，等）や入力値に依存するため，これらが異なると，1 回の実行で充放電する負荷容量も変わる．この値を正確に求めるためには，演算器を設計し，与えられた入力に対してどのノードがスイッチしたかを調べる必要があり，容易ではない．そのため，本節の評価では，演算の種類や入力値の違いで発生する負荷容量の変化は考慮しない．

7.2 評価結果

図 21 と**図 22** に，それぞれ，各ベンチマークの IPC と EDP を示す．6 本で組になっている棒グラフは，左から順に，(5f/1s)，(4f/2s)，(3f/3s)，(2f/4s)，(1f/5s)，(0f/6s) モデルの場合である．図 21 の縦軸は，(6f/0s) モデル（すべての iALU が高速なモデル）の IPC で正規化した IPC を示し，図 22 の縦軸は，(6f/0s) モデルの EDP で正規化した EDP を示す．

図 21，図 22 より，どのベンチマークにおいても，ほぼ同様の傾向を示すことが分かる．高速な iALU の数を減少させると，ほとんどの場合，EDP は単調に減少していく．しかし，提案手法は予測スラックに基づいて命令をスケジューリングするため，IPC の減少を抑制できている．(0f/6s) モデル（すべての iALU が低速なモデル）は，平均で IPC の低下が 20.2%，EDP の削減率が 41.6%となる．これに対し，(1f/5s) モデルは，EDP の削減率が 34.5%もあるにもかかわらず，IPC の低下を 10.5%まで改善することができる．また，(3f/3s) モデルは，IPC の低下がわずか 3.8%であるにもかかわらず，EDP の削減率が 20.3%にもなる．

文献 6) では，本研究とはベンチマーク・プログラムやプロセッサ構成が異なるものの，スラック予測機構として従来手法を用いて，(3f/3s) モデルを評価している．その結果，4.5%の IPC の低下で，19%の EDP を削減できることを示している．これより，提案手法は，従来手法と同様の結果を示すことが分かる．

以上の評価では，機能ユニットの消費電力にのみ着目している．スラック表の消費電力も考慮すると，プロセッサ全体の消費電力は低下しない可能性が十分あり，今後の課題である．しかし，現状においても，機能ユニットの消費電力を抑えることで，チップ上のホットスポットを削減できるという効果が得られると考える．

7.3 $Vmax$ を 2 以上とした場合の応用例

前節で評価した応用例では，各命令を実行する際の緊急度を 3 種類以上に分けることができるというスラックの利点が活かされていない．そこで，提案するスラック予測機構において，$Vmax$ を 2 以上とした場合の応用例を示す．

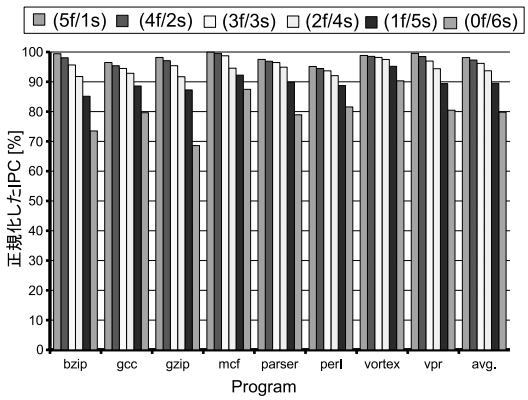

図 21 IPC
Fig. 21 IPC.

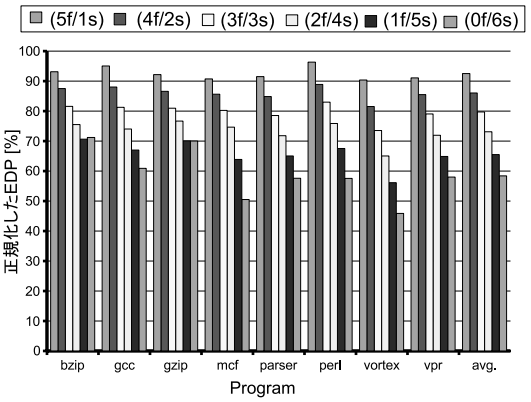

図 22 EDP
Fig. 22 EDP.

応用例として，機能ユニットの低消費電力化を行ったプロセッサの性能低下を抑制するいうものが考えられる．たとえば，$Vmax = 2$ とした (3f/3s) モデルにおいて，命令スケジューラが選択した命令を以下のように iALU に割り当てる．まず，予測スラックが 0 である命令を高速な iALU に割り当てる．高速な iALU に空きがない場合は，低速な iALU に割り当てる．次に，予測スラックが 2 である命令を低速な iALU に割り当てる．低速な iALU に空きがない場合は，高速な iALU に割り当てる．最後に，予測スラックが 1 である命令を低速な iALU に割り当てる．低速な iALU に空きがない場合は，高速な iALU に割り当てる．これにより，予測スラックが 1，あるいは，2 である命令の総数が，低速な iALU の数を上回った場合に，緊急度がより高い（予測スラックが 1 である）命令を優先的に高速な iALU に割り当てることができる．

上記以外にも，予測スラックを基に命令スケジューリングを行い，性能向上をはかるという応用例が考えられる．たとえば，$Vmax = 2$ とした (3f/3s) モデルにおいて，命令スケジューラに次の修正を加える：オペランドの揃った命令から，予測スラックの小さい順に命令を選択し，選択しなかった命令の予測スラックが 1 か 2 であれば，それを 1 減らす．なお，選択されなかった命令の予測スラックを 1 減らすのは，選択されないことで，その命令の実行開始が 1 サイクル遅れるからである．この修正により，予測スラックが n である命令の代わりに，予測スラックが $n+1$ 以上である命令を選択してしまうということがなくなる．その結果，緊急度に応じた順番で，命令を実行できるようになるため，低消費電力化による性能低下を緩和できる可能性がある．

8. 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 15 の場合，わずか 2.5%の IPC 低下で，31.6%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ことが分かった．また，機能ユニットの低消費電力化を行った場合，わずか 3.8%の IPC 低下で，EDP を 20.3%削減できることが分かった．

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．

参 考 文 献

1) Burger, D., 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 (Aug. 2002).
3) Fields, B., et al.: Focusing Processor Policies via Critical-Path Prediction, *Proc. ISCA-28* (June 2001).
4) Fields, B., et al.: Using Interaction Costs for Microarchitectural Bottleneck Analysis, *Proc. MICRO-36* (Dec. 2003).
5) Fields, B., et al.: Slack: Maximizing Performance under Technological Constraints, *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, (May. 2005).
7) Hennessy, J.L., et al.: *Computer Architecture: A Quantitative Approach, 2nd Edition*, Morgan Kaufmann Publishing Inc., San Francisco, CA (1996).
8) 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 (June 2001).
9) Levy, M.: Samsung Twists ARM Past 1 GHz, *Microprocessor Report 2002-10-16* (Oct. 2002).
10) 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 (May 2004).
11) Seng, J.S., et al.: Reducing Power with Dynamic Critical Path Information, *Proc. MICRO-34* (Dec. 2001).
12) Shivakumar, P., et al.: CACTI 3.0: An Integrated Cache Timing and Power, and Area Model, Compaq WRL Report 2001/2 (Aug. 2001).
13) Tune, E., et al.: Dynamic Prediction of Critical Path Instructions, *Proc. HPCA-7* (Jan. 2001).

(平成 18 年 1 月 27 日受付)
(平成 18 年 6 月 5 日採録)

小林良太郎（正会員）

1995 年名古屋大学工学部電子情報学科卒業．1997 年名古屋大学大学院工学研究科電子情報学専攻博士課程前期課程修了．2000 年名古屋大学大学院工学研究科電子情報学専攻博士課程後期課程満了．工学博士．2000 年名古屋大学大学院工学研究科電子情報学専攻助手．1999 年情報処理学会山下記念研究賞受賞．2002 年情報処理学会論文賞受賞．計算機アーキテクチャの研究に従事．

林　久紘

2004 年名古屋大学工学部電気電子情報工学科卒業．2006 年名古屋大学大学院工学研究科電子情報システム専攻博士課程前期課程修了．同年（株）日立製作所に入社．

島田　俊夫（正会員）

1968 年東京大学工学部計数工学科卒業．1970 年東京大学大学院修士課程修了．同年電子技術総合研究所入所．1993 年より名古屋大学大学院工学研究科電子情報学専攻教授．人工知能向き言語，LISP マシン，データフロー計算機の研究に従事．最近はマイクロプロセッサのアーキテクチャやチップ内並列処理の研究を行っている．1988 年度市村賞，1994 年度情報処理学会論文賞，1995 年注目発明，2002 年度情報処理学会論文賞受賞．工学博士．