実行パス履歴情報を利用した分岐予測手法

石井康雄[†] 平木敬[††]

本稿では新しい分岐予測手法である，Path Trace Branch Prediction（PTBP）を提案する．既知のパーセプトロンモデルの分岐予測器は，部分予測のパス情報として 1 度にたかだか 2 つの分岐命令のアドレスしか利用していない．PTBP では i 番目の部分予測値に i 個前の分岐命令からターゲットの分岐命令までに通過した i 分岐命令のアドレスを多数利用した Hash 値を用いる．PTBP ではこの Hash 値を詳細な実行パス履歴情報として活用し，従来手法と比較して高い分岐予測精度を実現する．また，PTBP はパイプライン化により分岐予測のレイテンシを 2BC-Gskew 予測器と同程度にできる．PTBP の評価をプロセッサシミュレータを用いて行った．その結果 8 KB 構成のパイプライン化した PTBP では既知の分岐予測器と比較して分岐予測ミス率が 8.8%削減され，IPC が 1.9%向上した．

Path Trace Branch Prediction

Yasuo Ishii[†] and Kei Hiraki[††]

In this paper, we propose Path Trace Branch Prediction (PTBP) which improves conventional branch predictors based on perceptron model. In conventional branch predictor based on the model, neurons are made from the limited path information that involve at most 2 addresses of conditional branch instructions. PTBP improves prediction accuracy by using a detailed path information for i_{th} neuron generation. This information is made from hash values that involve all of the last i addresses of conditional branch instructions. PTBP can be configured latency-sensitive by ahead pipelining. The latency of pipelined PTBP is comparable to that of the 2BC-Gskew branch predictor. We evaluate PTBP by the simulation. Its result shows that 4-lined pipelined PTBP with an 8 KB hardware budget reduces the miss prediction rate of 8.8% and improves IPC by 1.9% from Piecewise Linear Branch Predictor.

1. はじめに

マイクロプロセッサでは，命令間の制御依存がパイプライン実行を妨げることのないように分岐予測が採用される．近年，命令パイプライン長の深化により，分岐予測ミスペナルティは増加傾向にある．ミスペナルティの実行性能への影響を軽減するため精度の高い分岐予測手法を採用することはプロセッサの性能向上に不可欠であり，現在までさまざまな手法が提案されてきた．

その手法の 1 つとして Neural Network の簡単なモデルであるパーセプトロン[1] を利用した分岐予測器が Jiménez ら[7] によって提案された．パーセプトロンモデルを利用した分岐予測手法では，従来の Gshare 予測器[11] や 2 レベル分岐予測器[17] と比較して長い履歴長が利用できるため，高い予測精度が実現できる．しかし，パーセプトロンモデルにはいくつか問題点がある．まず，従来のパーセプトロンモデルの分岐予測器では，分岐命令の特性として内積演算による線形分離が不可能な場合には根本的に予測が不可能になる[5]．また，パーセプトロンでは分岐予測結果の計算の際に多くの整数値の加算をする必要があり，この加算のレイテンシが実行性能の向上を妨げる[4]．レイテンシの影響を隠蔽するためにはパイプライン化が必須となる．しかし，従来手法で深いパイプライン化を施すとターゲットの分岐命令から遠い段階で分岐予測を開始し，分岐予測精度に悪影響を与える．

本稿では，従来のパーセプトロンモデルの問題点を解決した新しい分岐予測手法，Path Trace Branch Prediction（PTBP）を提案する．PTBP は既存の実行パス履歴を利用する分岐予測器を発展させた分岐予測手法である．この手法では分岐予測に従来のパーセ

† 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Computer Science, Graduate School of Information Science and Technology, The University of Tokyo
†† 東京大学大学院情報理工学系研究科創造情報学専攻
Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo

プトロンモデルで利用していたパス情報に追加して，さらに詳しい実行パスの履歴情報を利用する．実行パス上の分岐命令のアドレスを多数利用した Hash 値を用いる．この詳細な実行パス履歴情報を利用することにより，従来のパーセプトロンモデルの分岐予測器の問題点を解決する．パーセプトロンの重み値の計算に実行時に通過したパスの情報を追加することにより，従来手法では線形分離が不可能だった問題に対して正しい予測を行える．また，PTBP ではパイプライン化を施すことにより 2BC-Gskew 予測器[15] のような 2 ビットカウンタを利用したハイブリッド型の分岐予測器と同程度のレイテンシで分岐予測が行える．

本稿の構成は以下のとおりである．2 章でパーセプトロンを利用した既知の分岐予測器について述べる．3 章では提案手法である Path Trace Branch Prediction を説明し，4 章でその評価を行う．5 章でまとめる．

2. 関 連 研 究

2.1 パーセプトロンモデルの分岐予測器

Neural Network モデルを利用した分岐予測器としてはパーセプトロン[1] を利用したモデルが知られている．パーセプトロンは人工知能の分野で古くから利用されている Neural Network のモデルの 1 つである．

このモデルを利用した分岐予測器では Gshare 予測器や 2 レベル分岐予測器と同じくグローバル分岐履歴を予測に用いる．予測の際には得られた分岐予測の 1 ビットずつに対して重み付けをする．この重み付けは命令ごとや実行パスごとに固有の値を利用する．得られた重み値は関連付けられた分岐履歴の **Taken**/**NotTaken** の値に従って符号を変化させ，部分予測値を生成する．この部分予測値の総和により最終的な予測結果を決定する．分岐命令の終了時には重み値を実行結果により更新し，学習をする．

パーセプトロンモデルを利用した分岐予測器の特徴は必要な記憶領域が履歴長に対して線形となることである．それに対してパターン履歴表を用いる予測器では必要な記憶領域は履歴長に対して指数関数的に増加する．したがって，パーセプトロンモデルを利用することにより従来の予測器では実現できなかった長い履歴長を活用した高精度の分岐予測ができる．その予測精度の高さは 2004 年 12 月に開催された Championship Branch Prediction（CBP）で決勝に進んだ 6 手法のうち 4 手法がパーセプトロンモデルを直接利用した分岐予測手法であり，その他の 1 手法も Neural Network モデルを活用し，パーセプトロンモデルに近いモデルを利用していたことから確認できる[2),3),5),10),12),13)]．

パーセプトロンモデルの分岐予測器は Fusion 型ハイブリッド[16] といわれるハイブリッド化手法と相性が良く CBP でも Desmet[2]，および，Loh[10] が 2BC-Gskew とパーセプトロンモデルの分岐予測器のハイブリッド化手法を提案し，高い予測精度を実現した．しかし，これらの手法は分岐予測の生成にかかるレイテンシが大きくプロセッサ上に実装するのは現実的でない．

以下では，すでに提案されている構成を述べる．

2.1.1 Global/Local Perceptron[7]

Global/Local Perceptron はパーセプトロンモデルを用いた予測器の提案論文において提案された．

この手法では，パーセプトロンの重み値をターゲットの分岐命令と関連付ける．この関連付けられた重み値と分岐履歴の内積により分岐予測を行う．分岐履歴は従来の 2 レベル分岐予測器と同様の手法で収集する．この手法で用いる分岐履歴は PAp などで利用されるローカル分岐履歴，および，GAp などで利用されるグローバル分岐履歴の両方を活用する．ローカル履歴を使わない場合は，単に Global Perceptron といわれる．

Global/Local Perceptron は高い分岐予測精度を実現するが，内積演算の処理が大きいため，ターゲットの分岐命令のアドレスが与えられてから分岐予測結果が得られるまでのレイテンシが大きくる．このため，プロセッサ上に実装することは現実的でない．

2.1.2 Path-Based Perceptron[4]

Path-Based Perceptron では，予測にターゲットの分岐命令の直前 h 個の分岐命令のアドレス情報を用いる．過去 i 番目の分岐履歴と関連する重みの値を当時の分岐命令のアドレスを利用したインデクスで読み出し，パーセプトロンの内積演算を行う．この手法では過去の分岐命令のアドレスを限定されたパス情報として利用する．

また，**図 1** に示した構造を用いることで効率的にパイプライン化を行うことが可能である．この演算はすべての重みテーブルで過去に実行した命令のアドレス $A[i]$ を利用して重み値 $W[A[i], i]$ の読み出しを行う．読み出された値は対応する分岐履歴の値 $H[i]$ に応じて符号を操作する．その値と前のパイプラインレジスタの値の加算結果を次のパイプラインレジスタに格納する．パイプラインの終端では Table 0 の値を加算し，最終結果として出力する．

このパイプラインのクリティカルパスは Table 0 の値の読み出しと図 1 (A) の加算の総和となる．このレイテンシは 2 ビットカウンタを利用する 2 レベル分岐

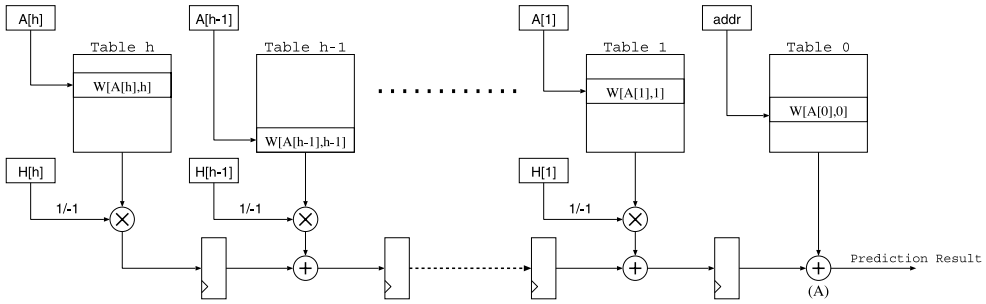

図 1 Path-Based Perceptron
Fig. 1 Path-Based Perceptron.

予測器と同程度で実装できる．

この手法では Table i から生成される予測値は i 分岐命令前に読み込まれた部分予測結果である．仮に途中の i 命令間で，この部分予測と異なる実行パスをたどった場合には，その値の持つ偏向情報の意味がなくなる．したがって，分岐履歴長を伸ばしすぎると，予測開始地点とターゲットの命令の時間距離が増し，予測と異なる実行パスをたどる可能性が指数関数的に増加し，予測精度に悪影響を与える．この特徴のために，利用できる資源が増えた際にも，他方式と比較して履歴長が長くできず，予測精度が向上しにくい．なお，本稿ではこの構成を Path-Base 型と呼ぶ．

2.1.3 Piecewise Linear 分岐予測器[6)]

この手法は Championship Branch Prediction で 3 位となった Idealized Piecewise Linear Branch Predictor[5)] をパイプライン化した予測器である．Idealized Piecewise Linear Branch Predictor は i 個前の条件分岐命令のアドレスとターゲットの分岐命令のアドレスからテーブルのインデクスを生成する方式で他方式よりも高い精度を持つ．Path-Base 型と Global Perceptron の両方の特性を兼ね備えており，両方式の欠点を補完する分岐予測方式である．

しかし，Idealized Piecewise Linear Branch Predictor をパイプライン化する際には，途中経過のデータ量が膨大になり，その保存のための資源量が膨大になる．たとえば，Piecewise Linear 分岐予測器の回路規模は 256 KB 構成で 408 個の 10 ビット加算器が必要となり，回路の複雑性が増し，発熱や消費電力が増加する．

3. Path Trace Branch Prediction

Path-Base 型では 2.1.2 項で示したとおり長い履歴長が用いることができないため，高い予測精度を実現することができない．この問題は i 番目の履歴と関連するテーブルのインデクスが i 命令前の分岐命令のアドレスのみとなっていることに起因する．この問題を**図 2** に示した．図 2 では B_0 がターゲットの分岐命令を含む基本ブロックであるとする．

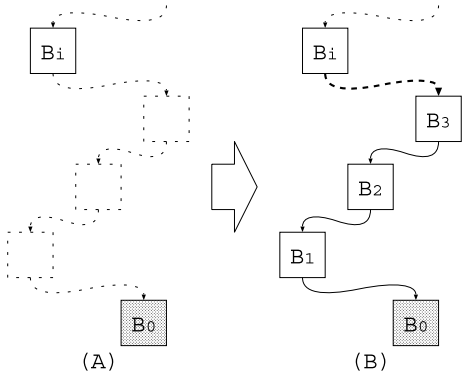

図 2 Path-Base 型 (A) と Path Trace (B)
Fig. 2 (A) Path-Based and (B) Path Trace information.

パーセプトロンモデルの分岐予測器では分岐命令（B_0）の予測を行うために，図 1 のように，Table 0 から Table h までの h+1 個のテーブルから重み値を読み出し，それを分岐履歴に従い符合反転した値を h+1 個生成する．予測結果は，この値の総和により求められる．ここでは，この h+1 個の値を部分予測結果と呼ぶ．最終結果は，この部分予測結果から計算されるため，この部分予測結果の精度はパーセプトロンモデルの分岐予測器にとって，最も重要な要素である．

Path-Base 型では図 1 に示すように，各テーブルのインデクスは過去に実行した分岐命令のアドレスそのものである．つまり，ターゲットの分岐命令（B_0）の予測を行うための部分予測結果を 1 つのアドレスから決定している．このとき部分予測結果の生成には図 2 (A) のように B_1 から B_{i-1} の情報をまったく利用していない．

この特徴のため，分岐履歴長を一定以上に伸ばすと i の値が大きくなることがある．B_i と B_0 の距離が大きいと B_i での予測は古い情報を用いて生成される

ため，その部分予測結果は B_0 に対して正確でない予測値を返す可能性が高まる．以上より，履歴長を伸ばしすぎると，古い分岐履歴情報のみを用いる部分予測結果が単なるノイズとなる可能性が増え，最終的な分岐予測の精度にも悪影響を与える．

これを解決するために，Piecewise Linear 分岐予測器では Path-Base 型のパス情報に加えターゲットの分岐命令 B_0 のアドレスを部分予測結果の生成に含め，その予測精度を高めた．しかし，完全に問題が解決されたとはいえない．

この問題を解決し，より長い履歴長でより高い予測精度を実現する Path Trace Branch Prediction（PTBP）を提案する．この手法では i 番目の履歴と関連するテーブルのインデクスの計算に対してターゲットの分岐命令から遡って i 個目の条件分岐命令までのパス情報を利用する．つまり，図 2 (B) のように B_0 の予測を行うために B_0 から B_i の間に通過した実行パス上のすべてのアドレスを詳細な実行パス履歴情報として利用する．この情報を分岐予測に利用することにより Path-Base 型の問題点を解決する．PTBP では部分予測に i 個前の分岐命令からターゲットの分岐命令までのパス情報を含むため，i 命令前の段階で得られた情報のみで部分予測を行う Path-Base 型と比較して高い予測精度を実現することができる．

この章では，まず分岐予測のレイテンシを無視した理想状態での分岐予測のアルゴリズムを述べ，その後にプロセッサ上に実装可能なパイプライン化を施した構成を述べる．

3.1 Idealized PTBP

分岐予測のアルゴリズムの大部分は従来と同じパーセプトロンモデルである．したがって，分岐履歴の各ビットに対してテーブルを 1 つ用意して分岐と関連した分岐の偏向を格納する．従来の Path-Base 型や Global/Local Perceptron との違いは各テーブルのインデクス計算の手法である．

3.1.1 分岐予測アルゴリズム

この分岐予測手法は従来の分岐予測手法と同じくターゲットの分岐を予測する関数と分岐命令の終了後に予測器を更新する手続きの 2 フェーズからなる．以下では，このアルゴリズムを構成する変数に関して説明する．

addr 分岐予測の対象となる分岐命令のアドレス．

h 分岐予測器で用いる履歴長．

θ 予測器の更新を行う際に，W を更新するかどうかを決定する定数．正の整数値である．

N 後述する 2 次元配列 W の行方向のエントリ数．

$H[1..h]$ 分岐予測器で用いるグローバル分岐履歴である．過去に実行した直前の h 分岐命令の **Taken**/**NotTaken** を格納する．シフトレジスタで構成される．分岐命令を実行したときには，その結果を $H[1]$ に挿入する．

$A[1..h]$ 実行された分岐命令のアドレスの配列である．過去に実行した直前の h 分岐命令のアドレスを格納する．したがって，$A[i]$ には，$H[i]$ を生成した分岐命令のアドレスが格納される．シフトレジスタで構成され，$H[1..h]$ と同時に更新される．更新時には $A[1]$ に新しいアドレスが挿入される．この情報を用いて実行パスの情報を生成する．

$W[0..N-1, 0..h]$ パーセプトロンの重みベクトルを格納するための 2 次元配列である．8 ビットの整数を格納する．1 番目のインデクスには関連する実行パスを基にした Hash 値を利用する．2 番目のインデクスには関連する分岐履歴 $H[i]$ のインデクスを用いる．

output 分岐予測結果を示す整数値である．*output* が負ならば **NotTaken**，それ以外ならば **Taken** が予測結果となる．

prediction 分岐予測の結果を示す真偽値である．

outcome 分岐命令の実行結果の **Taken**/**NotTaken** を示す真偽値である．

以上の変数を利用した Idealized PTBP の予測のアルゴリズムの擬似コードを**図 3** に，更新のアルゴリズムの擬似コードを**図 4** に示す．なお，本稿では $Rot(a, b)$ を a **を左方向に** b **ビット回転する**という意味で利用する．また，擬似コード上でのインデクスは

```
1  bool prediction(addr : integer)
2  begin
3    output := W[addr, 0]
4
5    for i in 1..h do
6      index := index xor rot(A[i], i)
7
8      if H[i] = Taken then
9        output := output + W[index, i]
10     else
11       output := output − W[index, i]
12     end if
13     ; 分岐履歴で符号転換し総和を求める
14   end for
15   prediction := output ≥ 0
16 end
```

図 3 Idealized PTBP の分岐予測アルゴリズム
Fig. 3 Prediction algorithm for the Idealized PTBP.

```
1  void update(outcome : bool, addr)
2  begin
3    if (output ≥ 0) ≠ outcome or
4       |output| < θ then
5       ; 分岐予測値により更新するかを決定
6
7       ; 以下では W を更新する
8       if outcome = Taken then
9         W[addr, 0] := W[addr, 0] + 1
10      else
11        W[addr, 0] := W[addr, 0] − 1
12      end if
13
14      for i in 1..h do
15        index := index xor rot(A[i], i)
16
17        if H[i] = outcome then
18          W[index, i] := W[index, i] + 1
19        else
20          W[index, i] := W[index, i] − 1
21        end if
22      end for
23      ; ここまでで W の更新を終了する
24    end if
25
26    ; 以下で分岐履歴情報を更新
27    H := (H << 1) or outcome
28    A := (A << 1) or addr
29 end
```

図 4 Idealized PTBP の更新アルゴリズム
Fig. 4 Update algorithm for the Idealized PTBP.

エントリ数の上限値で **mod** をとった値を暗黙的に利用する．たとえば，$W[addr, i]$ は $W[addr \ \mathbf{mod}\ N, i]$ を意味する．

Idealized PTBP と従来手法との相違点は重みベクトルのテーブル W に対するインデクスの計算手法である．たとえば，図 3 の 6 行目および図 4 の 15 行目の記述を $index := addr$ と変更すると Global Perceptron が実現ができ，同じ箇所を $index := A[i]$ と変更するとパイプライン化をしない Path-Base 型が実現できる．Piecewise 型は $N = N_1 \times N_2$ と分解できるとき $index := addr \times N_2 + A[i]$ とすると実現できる．ただし，$addr$ と $A[i]$ は N_1 と N_2 で **mod** をとった値であるとする．したがって，既存手法ではインデクス $index$ の計算にたかだか 2 つの分岐命令の情報しか含まれていない．それに対して Idealized PTBP では i 個前の分岐命令からターゲットの分岐命令の間に通過した条件分岐命令のアドレスをすべて用いている．つまり，i 番目の履歴に関連する値では i 個の分岐命令のアドレス情報を利用できる．

パス情報の生成は実行してきたパス上にある分岐命令のアドレスの Hash 値を用いて実現する．Idealized PTBP では Hash 関数としてパス情報を経路上のすべての分岐命令のアドレスのビット回転と XOR により実現した．これは図 3 の 6 行目および図 4 の 15 行目に対応する．以上より，すべての分岐命令のパス情報を分岐予測に対して利用することができる．なお，Hash 関数を XOR とビット回転で実現した理由は，XOR 演算はハードウェア実装を容易にするためであり，ビット回転演算は同じ分岐命令を連続して通過したときに XOR 演算の結果が 0 になることを防ぐためである．

テーブルから得られた整数値は分岐履歴の結果により必要に応じて符号を変化させた後に総和をとる．これは従来のパーセプトロンモデルの分岐予測器と同様である．符号は対応する分岐履歴が **Taken** であった場合には変化させず，**NotTaken** であった場合には正負を反転させる．これは図 3 の 8 行目から 12 行目に示した．

分岐予測器の更新は，従来のパーセプトロンモデルの分岐予測器と同様に，分岐予測の結果が間違っているか，得られた整数値の絶対値が閾値以下であるときに行う．更新する場合にはパーセプトロンの重みテーブル W の値の更新を行う．これは図 4 の 7 行目から 23 行目で示した．予測時に利用した分岐履歴 $H[i]$ が分岐の結果 $outcome$ と等しいときにはテーブルの値を $+1$ し，そうでない場合には -1 する．これにより，次回以降の予測では更新が反映された値が返される．なお，各テーブルは 8 ビットの整数値を保持する．

3.1.2 分岐予測の原理

従来の Path-Base 型のパーセプトロンモデルでは，i 番目の部分予測値に i 個前の分岐命令のアドレスを利用するのみであり，ターゲットの分岐命令が i 分岐前に予測した命令と異なっていたり，ターゲットの分岐命令までに通過する分岐命令数が変化し実行パスが変化したりすると，意味のない予測結果を返してしまう問題点があった．これは Piecewise Linear 分岐予測でも解決されておらず，高い予測精度を実現することの妨げとなる．

PTBP では，i 番目の部分予測値の生成に i 個前の分岐命令から現在の分岐命令までにどのパスを通過してきたかの情報が利用できる．この情報は従来手法では利用していない．通過したパスの情報が予測に含まれていれば，ターゲットの分岐命令までに通過した実行パスの変化が直接テーブルの読み出しのインデクスに反映される．したがって，パス情報を利用した PTBP では高い分岐予測成功率を示すことができる．

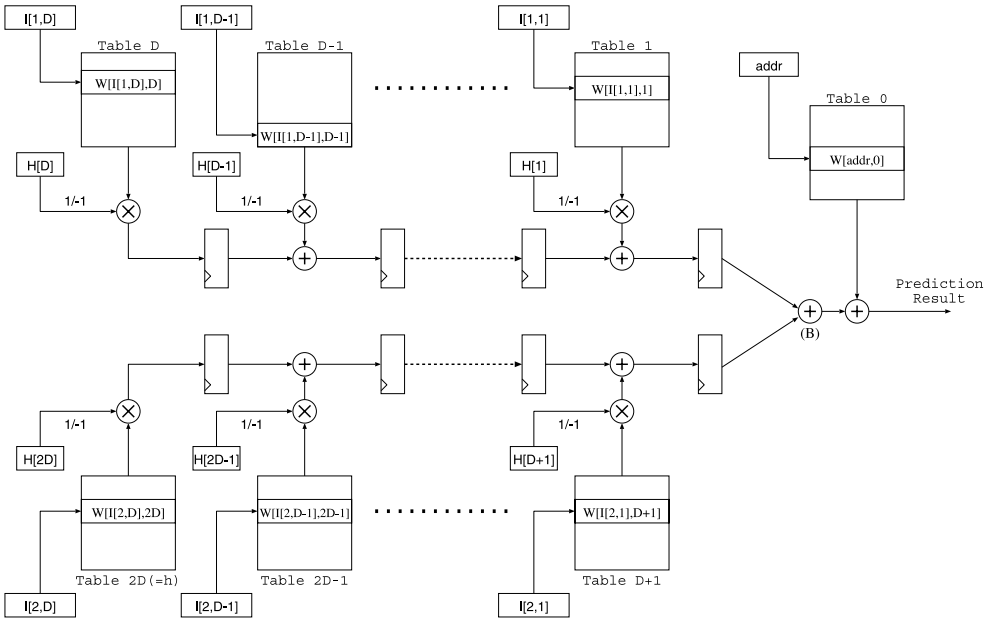

図 5 Pipelined Path Trace Branch Predictor の 2 本鎖の場合の構造
Fig. 5 Architecture of the Pipelined Path Trace Branch Predictor with 2 lines.

Idealized PTBP では詳細なパス情報を利用するため高い分岐予測精度が実現できる．しかし，利用する履歴長が h の場合に，ターゲットの分岐命令のアドレス *addr* が得られてから分岐予測結果が求められるまでに h 個の加算を含むため分岐予測のレイテンシが増大する．次節では Idealized PTBP のレイテンシに配慮し，パイプライン化を施した構成に関して述べる．

3.2 Pipelined PTBP

Idealized PTBP はその構造から分岐予測のレイテンシが非常に大きくなるため，プロセッサ上に実装することは現実的ではない．そこで，この節では Idealized PTBP をパイプライン化した Pipelined PTBP を提案し，分岐予測のレイテンシを 2BC-Gskew 予測器と同程度に抑える手法を述べる．

Pipelined PTBP では Path-Base 型では 1 本で行われていた演算パイプライン（演算鎖）を複数保持する．Path-Base 型と比較して M 本鎖構成の Pipelined PTBP はパイプライン段数を $1/M$ に削減する．これにより Path-Base 型の長いパイプライン長に起因した問題を解決する．Pipelined PTBP の構成例を**図 5** に示す．図 5 は簡単のため 2 本鎖の構成例を示したが，以下の議論では 4 本鎖から 8 本鎖程度のライン数を持つ構成を想定する．なお，3 本鎖以上の構成では図 5 中の各鎖の結果の総和をとる加算器（B）にキャリーセーブアダー（CSA）を用いた最適化を施す．

Pipelined PTBP では分岐予測値の計算をいくつかのパイプラインに分けることにより，予測開始から予測完了までの時間の短縮を実現する．それぞれのパイプライン上では Path-Base 型と同様の手法で予測のパイプライン化を行う．すなわち，各パイプラインは重みテーブル W，分岐履歴 H，重みテーブルのインデクス I を持つ．各テーブルの間にはパイプラインレジスタ R があり，途中経過をこのレジスタに格納する．パイプラインの深さが D と仮定をすると D ステップかけて部分予測値を生成し，その値を利用して最終的な分岐予測結果を求める．アルゴリズムの詳細に関しては次項以降に示す．

以下では分岐予測アルゴリズムと動作原理を説明し，レイテンシの検証を行う．

3.2.1 分岐予測アルゴリズム

ここでは Pipelined PTBP の分岐予測手法に関して述べる．この項では 3.1.1 項で利用した変数を同様の意味で利用するとともに，Pipelined PTBP を説明するためにいくつか新しい変数を追加する．それらの詳細を以下に示す．

- M Pipelined PTBP のパイプラインの本数．本稿では 4 から 8 程度を想定する．
- D Pipelined PTBP の合流地点（図 5 (B)）までのパイプラインの深さ．h，M が与えられた場合に $D = \lceil h/M \rceil$ で定義される．なお，h が M で割り切れない場合には，M 本目のパイプラインの深さ D' を $h - (M-1) \times D$ とする．D' を利

```
 1  bool prediction(addr : integer)                  Pipelined PTBP の分岐予測アルゴリズム
 2  begin
 3    output := W[addr, 0]                           分岐偏向の値の読み出し
 4    for i in 1..M do
 5      output := output + SR[i, 1]                  事前に計算していた結果を加算する
 6    end for
 7    prediction := (output ≥ 0)                     分岐予測結果の計算が終了
 8
 9    SH := (SH << 1) or prediction                  以下では次回以降の分岐予測のための計算を記述
10    for i in 1..M in parallel do
11      SI'[i, 1] := addr xor rot(SI[i − 1, D], 1)   SI の値を XOR を利用して更新する．SI[0, D] は 0
12      SI'[i, 2..D] := SI[i, 1..D − 1]              各パイプラインの SI の値をシフトする
13
14      for j in 1..D in parallel do                 D 命令先までの分岐予測の計算をする
15        k := (i − 1) × D + j                       どの SH を利用するかを計算する
16        if SH[(i − 1) × D] = Taken then            以下では SR の更新を行う
17          SR'[i, j] := SR[i, j + 1] + W[SI'[i, 1], k]   関連する分岐履歴が Taken の場合に実行
18        else
19          SR'[i, j] := SR[i, j + 1] − W[SI'[i, 1], k]   関連する分岐履歴が NotTaken の場合に実行
20        end if                                     ここまでで SR の更新を終了する
21      end for
22    end for
23    SR := SR'    SI := SI'                         分岐予測の途中結果を投機的に更新
24  end
```

図 6 Pipelined Path Trace Branch Predictor の予測アルゴリズム
Fig. 6 Prediction algorithm of the Pipelined Path Trace Branch Predictor.

用しても，アルゴリズムに本質的な違いはできない．したがって，以降では h は M で割り切れるとして議論を行う．

$SH[1..h]$　投機的に更新を行うグローバル分岐履歴．プロセッサの実行パイプラインが深い場合には，パイプラインの前半で実行される予測と終端で実行される更新の時間間隔が広がる．このときに，先行命令の終了を待たずに予測を行うために，h ビットのシフトレジスタを用いて投機的に更新するグローバル分岐履歴を利用する．
この情報は予測時に投機的に更新される．分岐予測ミスなどが発生した場合には破棄され，H の内容がコピーされる．

$I[1..M, 1..D]$　W を引くための，整数値の配列．PTBP のパス情報の Hash 値を保持する．この値は過去に通過した分岐命令のアドレス $A[1..h]$ のいくつかをビット回転した後に XOR した Hash 値で生成される．本稿では 20 ビットの整数値を利用する．

$SI[1..M, 1..D]$　投機的に更新する W を引くための整数値の配列．SH と同じく，この情報は予測時に投機的に更新される．この情報が破棄されるときには I の内容がコピーされる．

$R[1..M, 1..D]$　予測結果をパイプライン化して計算する際に途中経過を格納する整数の 2 次元配列．分岐予測器の更新時に，予測結果が正しかったかどうかの検証に利用される．各要素は W よりも，やや大きい値が格納できるようにする必要があり，本稿では 10 ビット整数とする．

$SR[1..M, 1..D]$　投機的に分岐予測結果をパイプライン化して計算する際に途中経過を格納する整数の 2 次元配列である．R と同じ構成をする．予測関数内で投機的に更新される．SI，SH と同じく，この情報は予測時に投機的に更新される．この情報が破棄されるときには R の内容がコピーされる．

これらの変数を利用し，Pipelined PTBP のアルゴリズムの詳細を述べる．Pipelined PTBP も Idealized PTBP 同様に予測関数と更新手続の 2 フェーズからなる．その予測関数の擬似コードを**図 6** に示し，更新手続の擬似コードを**図 7** に示す．

この Pipelined PTBP の予測と更新の手続きはそれぞれいくつかの部分に分解できる．図 6 の予測関数は以下の 2 つに分割できる．1 つめは，3 行目から 7 行目までの分岐予測の最終結果の生成で，2 つめは，9 行目以降の次回以降の分岐予測のための投機的な分岐履歴情報の更新部分である．同様に，図 7 の更新手続は以下の 3 つに分割できる．1 つめは，3 行目から

```
 1  void update(outcome : bool; addr : integer)
 2  begin
 3    output' := W[addr, 0]                          ここから分岐予測結果の再計算
 4    for i in 1..M do                               output' は分岐予測結果算出のための変数
 5      output' := output' + R[i, 1]                 事前に計算していた結果を加算する
 6    end for                                        ここまでで分岐予測結果の生成を終了する
 7
 8    if (output' ≥ 0) ≠ outcome or |output'| < θ then   output' 値から W を更新するか否かを決定する
 9      if outcome = Taken then                      分岐偏向の値を更新する
10        W[addr, 0] := W[addr, 0] + 1               分岐が Taken の場合
11      else
12        W[addr, 0] := W[addr, 0] − 1               分岐が NotTaken の場合
13      end if                                       分岐偏向の更新を終了する
14
15      for i in 1..M in parallel do
16        for j in 1..D in parallel do               重みベクトルのテーブルを更新
17          k := (i − 1) × D + j
18          if H[k] = outcome then                   Idealized PTBP と同様に W を Update する
19            W[I[i, j], k] := W[I[i, j], k] + 1     対応する履歴と結果が一致した場合
20          else
21            W[I[i, j], k] := W[I[i, j], k] − 1     対応する履歴と結果が一致しなかった場合
22          end if                                   ここまでで W の更新を終了する
23        end for                                    ここから先では次回以降の分岐のための情報を
24      end for                                      確定させるための計算を行う
25    end if
26
27    H := (H << 1) or outcome                       分岐履歴を更新
28    for i in 1..M in parallel do
29      I'[i, 1] := addr xor rot(I[i − 1, h], 1)     I の値を XOR を利用して更新する．I[0, D] は 0
30      I'[i, 2..D] := I[i, 1..D − 1]                各パイプラインの SI の値をシフトする
31
32      for j in 1..h in parallel do                 以下では次回の分岐のための途中経過を計算
33        k := (i − 1) × D + j
34        if H[(i − 1) × D] = Taken then             以下では R の更新を行う
35          R'[i, j] := R[i, j + 1] + W[I'[i, 1], k]   関連する分岐履歴が Taken の場合に実行
36        else
37          R'[i, j] := R[i, j + 1] − W[I'[i, 1], k]   履歴が NotTaken の場合に実行
38        end if                                     ここまでで R の更新を終了する
39      end for
40    end for
41    R := R'    I := I'                           分岐履歴情報の更新を終了する
42
43    if prediction ≠ outcome then                 分岐予測をミスした場合は状態復帰を行う
44      SR := R    SH := H    SI := I               確定済みの情報を投機情報にコピーする
45    end if
46  end
```

図 7 Pipelined Path Trace Branch Predictor の更新アルゴリズム
Fig. 7 Update algorithm of the Pipelined Path Trace Branch Predictor.

25 行目までの重みベクトルテーブルの更新部分で，2 つめは，27 行目から 41 行目までの分岐履歴情報の更新部分で，3 つめは，43 行目以降の投機情報の破棄を行う部分である．以下ではそれぞれの部分に関して順に説明をする．

Pipelined PTBP の予測関数の結果の生成に関しては，それまでに生成してきた投機的な途中結果の値 $SR[1..M, 1]$ と，分岐命令自体の偏向である $W[addr, 0]$ から最終的な予測結果を生成する．この部分はプロセッサのクリティカルパスとなりうるため，簡潔なアルゴリズムで計算を行う必要がある．Pipelined PTBP では SR から得られる 4 から 8 個程度の 10

ビット整数値の総和を CSA を用いて計算し，その値に分岐命令の偏向を加算し予測結果を求める．これは SR の総和演算のレイテンシが $W[addr, 0]$ の読み出しのレイテンシより短いならば，クリティカルパスに入らずに実装することができる．

次に，予測関数内の投機情報の更新部分と，更新手続内の分岐履歴情報の更新部分に関して述べる．これらの 2 つの部分は，投機的な情報であるか，確定した情報であるかの違いがあるのみで本質的な違いはないのであわせて解説する．投機的な分岐履歴情報の更新にはパイプラインのバックエンドで生成される $outcome$ を利用することができない．したがって，生成した予測結果 $prediction$ を用いて投機的分岐履歴情報を更新する．この部分では前半にインデクス I，SI を求めた後に後半部分で途中結果 R，SR の更新をする．インデクスの再計算部分では，各ラインの先頭（$I[1..M, 1]$，$SI[1..M, 1]$）のみが値の再計算をし（図 6 の 11 行目，図 7 の 29 行目），そのほかの値は単にシフトレジスタで移動をする（図 6 の 12 行目，図 7 の 30 行目）．このインデクスの計算手法が Pipelined PTBP の特徴の 1 つである．なお，インデクスの生成が 1 ビット回転と XOR で構成されている理由は Idealized PTBP と同様である．途中結果の更新部では，計算されたインデクスを利用して全重みテーブルで値を読み出し，全ラインで同時に途中結果をパイプライン上で 1 段進める（図 6 の 16 行目から 20 行目，図 7 の 34 行目から 38 行目）．

最後に，更新手続内の重みテーブルの更新と投機情報の破棄に関して述べる．重みテーブル W の更新アルゴリズムは，従来のパーセプトロン方式の更新手法と同様である．すなわち，予測結果が間違っているか，予測結果の整数値の絶対値が閾値以下の場合に更新をする．更新することが決まった場合には，その予測を生成するときに利用した W のエントリを H と $outcome$ の値に従って $+1/-1$ する．投機情報の破棄では，分岐予測の結果が間違っていた際に SH，SI，SR の値を破棄する．そのときに，確定した情報を利用して生成した H，I，R の値をコピーすることにより，以降の分岐予測が正しい分岐履歴情報を利用して行えるようにする．なお，図中の変数 $prediction$ は分岐予測時に得られた分岐予測結果である．また，この投機情報の破棄は BTB ミスなどが発生した際にも実行される．

以上の手法によって，履歴長 h で Path-Base 型では $h+1$ 段となっていたパイプラインの深さを，M 本鎖の Pipelined PTBP では $\lceil h/M \rceil + 1$ 段に削減できる．すべての部分予測にターゲットの分岐命令に近い分岐命令の情報が利用できるため，分岐予測精度を向上させることができる．

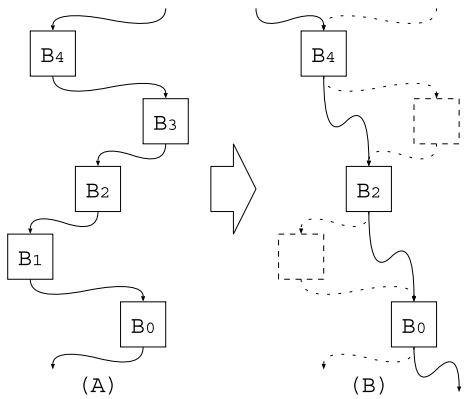

図 8 完全なパス情報 (A) と限定されたパス情報 (B)
Fig. 8 (A) Full Path Trace Information, (B) Limited Path Trace Information.

3.2.2 動作原理

Pipelined PTBP を Idealized PTBP と比較すると限定されたパス情報で予測を行っている．Idealized PTBP では，**図 8** の (A) のように実行パス上の全情報を利用する．それに対して Pipelined PTBP では図 8 の (B) のようにいくつか実行してきたパスの情報をとばす．これにより，パイプライン化を可能にする．図中の例は 1 分岐命令おきにパス情報を取得する例である．限定されたパス情報は完全なパスを反映した情報となるので，分岐予測においては高精度を実現する補助となる．Pipelined PTBP の実装では，パス情報は D 命令ごとに取得する．

パス情報を利用する分岐予測では，ターゲットとなる分岐命令の近くのパス情報を利用したほうが，その分岐命令の特性を反映した情報となるため，遠くの情報を利用するよりも分岐予測精度が高くなる．Path-Base 型では i 番目の部分予測値はその命令が実行された i 個後の分岐命令の分岐予測結果である．すなわち，Path-Base 型はターゲットの分岐命令から遠いうえに非常に限定されたパスを利用する手法であり，ターゲット分岐命令が得られてから分岐予測を行うのに比べて正確でない可能性が高い．

一方，Idealized PTBP のように完全なパス情報を利用する分岐予測手法では，ターゲットの分岐命令のアドレスが得られてからすべての計算を行うため分岐予測のレイテンシが増大する．このレイテンシの増大による性能低下で高い分岐予測精度を生かすことが不可能になる．

Pipelined PTBP は，この 2 つの相反する問題のト

レードオフの結果である．このことはPipelined PTBPは1本鎖構成にするとPath-Base型と同じ構造となり，履歴長がhのときにPipelined PTBPをh本鎖構成にするとIdealized PTBPと近い構造となることから確認できる．Pipelined PTBPでは，Path-Base型のようなパイプライン化をしたうえでターゲット分岐命令に近い地点のパス情報を得てから分岐予測を開始する．この特性により，高精度かつ低レイテンシな分岐予測を実現できる．

3.2.3 ハードウェア量の考察

Pipelined PTBPに必要なハードウェア量に関して考察する．図6，図7よりPipelined PTBPでは内部状態を保存するH，SH，R，SR，I，SI，WのRAMが必要となる．これらの容量は，履歴長h，重みテーブルのエントリ数N，パイプライン数Mとすると，HとSHはhビットのシフトレジスタ，RとSRはh要素の10ビット整数を格納するレジスタ，IとSIはh要素の20ビット整数を格納するレジスタ，Wは$h \times N$要素の8ビット整数を格納するRAMとなる．

記憶素子以外の組合せ回路では図5に示されるように，RやIを計算するために加算器とXOR回路が必要となる．I，SIの計算にはM個の20ビットのXOR回路が必要となる．R，SRの計算においては符号反転のためのXOR回路，および，10ビットの加算器が$h - M$個必要となる．各パイプライン演算の総和を計算する部分（図5(B)）でM個の10ビット値の総和演算が必要になる．

以下では，このハードウェア量を他方式のパーセプトロンモデルの分岐予測器と比較する．

同一の履歴長hとエントリ数Nを用いるPath-Base型の予測器と比較すると，I，SI計算のためのXOR回路が追加された形となる．このXOR回路の大きさはWに用いられるRAMや加算器などの部品と比較して非常に小さいため，Pipelined PTBPはPath-Base型に対してわずかなハードウェアを追加することにより実装可能である．よって，実現可能性は高いといえる．

また，既存手法で最も高い予測精度を示すPiecewise Linear分岐予測器と比較する．この手法では256 KB構成時に408個の10ビット加算器が必要になる[6]．これを同規模の構成のPipelined PTBPと比較をすると，Pipelined PTBPで必要な加算器の個数が51個であるため，加算器の個数が1/8になる．加算器は発熱の大きい部品であり，発熱や消費電力の点で効率化がなされることが予想される．

3.2.4 分岐予測レイテンシの考察

分岐予測のレイテンシに関して考察する．4本から8本鎖の構成のPipelined PTBPを想定する．Path-Base型では分岐命令のアドレスが得られてから分岐予測結果を得るまでに必要な資源は図1で確認できるとおり，(1) 途中経過を格納したパイプラインレジスタ，(2) 図1のTable 0の偏向格納テーブル，(3) 図1の(A)の加算器，の3つの部品である．クリティカルパスはTable 0の読み出しのレイテンシと図1の(A)の加算器によるレイテンシの総和となる．Table 0が0.5 KB程度の小さいRAMを採用していること，および，(A)の行う加算が10ビットと小さいことから，低レイテンシが実現される．そのレイテンシは2BC-Gskewと同程度になる[4]．

Pipelined PTBPをPath-Base型と比較すると，分岐命令のアドレスが得られてから分岐予測結果を得るまでに必要なハードウェア資源としては，図5の(B)の加算器が増えているのみである．ここでは，CSAを利用した回路の最適化により，4個から8個程度の10ビットの加算がTable 0に対するアクセス時間内に終了すると仮定する．(B)の加算のレイテンシはTable 0の読み出し時間とオーバラップさせることにより隠蔽できる．したがって，Path-Base型からのレイテンシの増加なくPipelined PTBPを実装できる．以上より，Pipelined PTBPは2BC-Gskewなどと同程度のレイテンシで予測結果を計算することができる．

また，各テーブルの読み出し部分のパイプライン化[14]を施すことで，より小さなアクセスレイテンシにすることができる．その場合は合流地点の加算の直前のパイプラインレジスタの位置を加算の直後にずらすなどのパイプライン構成の変更が考えられる．仮にパイプライン上でのパイプラインレジスタの位置を変更させても分岐予測のアルゴリズムに影響はないので，本稿ではこれ以上の言及はしない．

4. 評　　価

4.1 分岐予測精度の評価

この章ではPTBPの分岐予測精度をシミュレーションにより評価をする．提案手法と比較評価をするために，Global/Local Perceptron予測器，Path-Based Perceptron予測器，および，Piecewise Linear分岐予測器を既知のパーセプトロン型分岐予測器として評価対象とした．古典的な2ビットカウンタを用いた予測器としてGshare予測器，2BC-Gskew予測器を評価対象とした．

PTBPの構成に関しては分岐履歴長のほかに，パ

表 1 最適化した各予測器の記憶容量別の履歴長
Table 1 Tuned history length for each predictor.

Budget Size	Gshare	2BC-Gskew	Global/Local	Path-Based	Piecewise	PTBP
4 KB	14	24	34/12	19	19	19
8 KB	15	32	34/12	31	20	22
16 KB	16	32	38/14	33	24	26
32 KB	17	32	40/14	33	26	32
64 KB	18	40	50/18	36	43	43
128 KB	19	40	60/20	39	50	50
256 KB	20	40	70/20	41	51	51

イプライン本数が構成上のトレードオフとなる．今回のシミュレーションでは Pipelined PTBP に関しては 4 本鎖，8 本鎖の構成に関して評価を行った．Limit Study として Idealized PTBP に関して評価を行った．

解析に利用した実験環境を以下に示す．シミュレーションは SimpleScalar 3.0d を用いて作成した Trace を利用した．評価対象のプログラムは SPEC CINT2000 を利用した．

各構成で各プログラムに対して 4G 命令ずつ実行し，分岐予測ミス回数を測定した．分岐予測ミス回数から平均の分岐予測ミス率を計算し，その値で解析を行った．

評価する分岐予測器の構成は使用可能な記憶容量の大きさを 4 KB から 256 KB まで変化させる．それぞれの記憶領域の大きさ別の履歴長の値を**表 1** に記した．これらの値はそれぞれ提案論文中でのチューニングの結果[4),6),8)] から決定した．

2 章で示したとおり，Path-Base 型では記憶容量を大きくしても履歴長を大幅に伸ばすことができないため，64 KB よりも大きい構成ではパーセプトロンモデルの予測器の中で最も履歴長が短くなっている．また，32 KB 構成まで PTBP が Path-Base 型よりも履歴長が短いのは，テーブル W のエントリ数 N が一定数以上ない場合に，PTBP のテーブルの読み出しの Hash 値が競合するためである．Gshare 方式などの従来手法と同様にエントリの破壊的競合が発生すると予測精度に悪影響を与えるため[9)]，PTBP は小規模な構成では履歴長が伸ばせない．

パーセプトロンモデルの分岐予測器の更新時の閾値は $\theta = \lfloor 2.14 \times (h+1) + 20.58 \rfloor$ を利用した[6)]．

4.1.1 評価結果

各記憶容量ごとの平均のミス率をグラフにしたものを**図 9** に示す．グラフ上では既存手法として Gshare が Gshare 予測器，2BC-Gskew が 2BC-Gskew 予測器 Perceptron が Global/Local Perceptron 予測器，Path-Base が Path-Based Perceptron 予測器，Piecewise が Piecewise Linear 分岐予測器を示している．また，提案手法としては，Ideal が Idealized PTBP の結果を示し，Pipe4，Pipe8 がそれぞれ 4 本鎖，8 本鎖の Pipelined PTBP の結果を示している．

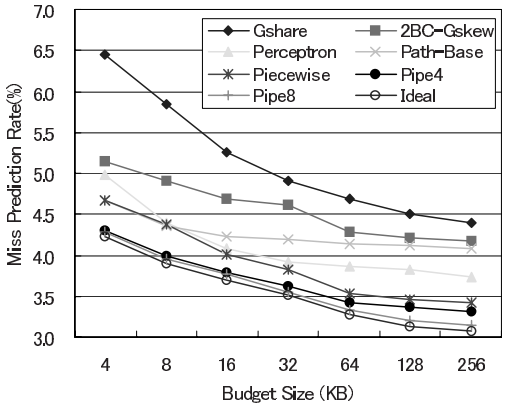

図 9 各分岐予測器の分岐予測ミス率
Fig. 9 Miss prediction rate of each predictor.

評価結果では Global/Local Perceptron 方式が 16 KB より大きい構成で Path-Base 型を上回る結果となった．これは文献 4) の実験結果と一致しない．この原因は分岐予測の実行命令数が文献 4) では 500M 命令なのに対して，我々の行ったシミュレーションでは 4G 命令実行していることに原因があると考えられる．一般に分岐予測では，分岐履歴長が伸びるほど分岐偏向の学習に時間がかかる．Global/Local Perceptron は今回測定したすべての分岐予測器の中で最も履歴長が長く，学習時間も長くなる．したがって，本稿の実験では文献 4) よりも 8 倍の命令数を実行したため，学習後の予測精度が結果に大きく影響し，今回のような結果になったと考えられる．

PTBP の予測精度に関して考察をする．全構成で PTBP が従来手法の予測精度を上回る結果が得られた．特に，一般的な構成である 8 KB 構成の分岐予測器においては Pipelined PTBP は Piecewise Linear 分岐予測器から分岐予測ミスを 4 本鎖構成で 8.8%，8 本鎖構成で 9.3%削減することが分かった．結果から 8 KB 構成の PTBP では履歴長が他方式と比較して

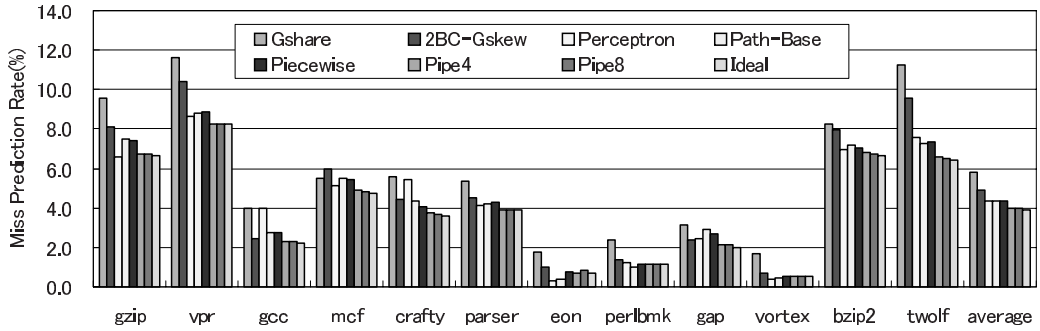

図 10 8 KB 構成での各ベンチマークでの予測精度
Fig. 10 Miss prediction rate of each predictor with 8 KB budget.

短いが，予測精度が高いことが確認できる．これは部分予測値の生成に詳細な実行パス履歴情報を用いたため，各部分予測値の精度が向上したことが原因と考えられる．実験結果から PTBP は同程度のハードウェア資源をより効果的に使うことのできる分岐予測手法であるといえる．

図 10 に各ベンチマークでの予測精度を示す．12 プログラム中 8 プログラムで，既知の分岐予測器で最高の予測精度を示すことが確認できる．以下では，いくつかのベンチマークの結果に関して考察を行う．

164.gzip と 252.eon では Global/Local Perceptron が最も良い結果なのは，評価対象の分岐予測器の中で，最も長い分岐履歴を利用しているため，ループの予測精度が高まっているためと考えられる．次に，176.gcc のベンチマークでは 2 ビットカウンタを利用する 2BC-Gskew が相対的に高い精度を示している．したがって，これはパーセプトロンモデルの分岐予測器が苦手なベンチマークであるといえる．しかし，PTBP はそのようなベンチマークでも高い予測精度を実現している．最後に，252.eon では 4 本鎖 PTBP が 8 本鎖 PTBP の性能を上回っている．これは W のインデクスがエントリ不足のため競合していることが原因であると考えられる．しかし，この影響での性能劣化は 0.1%と小さい．

以上より，PTBP は従来のパーセプトロンモデルの苦手としたプログラムの分岐命令の予測精度を高め，プロセッサの性能向上に貢献できる分岐手法であるといえる．

4.2 分岐予測器の実行性能への影響の評価

分岐予測精度とプロセッサ自身の実行性能の影響を調べるために SimpleScalar 3.0d Sim-Outorder に各分岐予測器を実装しその性能の比較を行った．

今回の評価では提案手法の比較対象として 2BC-Gskew，Path-Based Perceptron 予測器，Piecewise Linear 分岐予測器を利用した．PTBP は前節での実験と同じく 4 本鎖，8 本鎖，Idealized の 3 つの構成を評価した．

表 2 プロセッサの構成
Table 2 Architectual parameters.

Parameter	Configuration
Pipeline Depth	20
Issue Size	4-way
L1 I-cache	64 KB, 32 B blocks, 2-way
L1 D-cache	64 KB, 32 B blocks, 4-way
L1 D-cache Latency	2 cycles
L2 Cache	2 MB, 128 B blocks, 8-way
L2 Cache Latency	12 cycles
L2 Miss Latency	200 cycles
Branch Target Buffer	4096 entries, 4-way
ROB/LSQ entries	256 / 128

今回の評価に用いたプロセッサの構成を**表 2** に示す．20 段構成のパイプラインを持つプロセッサは，Intel Pentium 4 のパイプラインの深さが 20 段から 31 段であり，現代のプロセッサとしては適当である．各分岐予測器の構成に関しては前節と同じく表 1 に示した値を利用した．

評価対象のプログラムとしては SPEC CINT2000 を利用した．各ベンチマークに対して 256M 命令ずつ実行し，分岐予測精度とクロックサイクルあたりの実行命令数（IPC）を測定し，評価した．

4.2.1 評価結果

図 11 に各構成ごとの平均の分岐予測ミス率を示した．この結果は前節の結果と比較して相対的に 2BC-Gskew の予測精度が高くなっている．原因として考えられることは，学習速度の差である．2BC-Gskew は 2 ビットカウンタを利用するため学習速度がパーセプトロンモデルと比較して速い．今回のシミュレーションは実行命令数が 256M 命令と少なく，学習速度が結果に与える影響が大きい．したがって，2BC-Gskew の結果が前節と比較して相対的に良い予測精度を示し

ていると考えられる．しかし，依然として PTBP は全構成で他方式の予測精度を上回っており，その予測精度の高さが確認できる．

また，**図 12** に各構成ごとの IPC を示した．この結果は図 11 の結果を強く反映しており，高い予測精度を実現することがプロセッサの実行性能を向上させていることが確認できる．特に，最も一般的な 8 KB 構成の分岐予測器に関しては，Piecewise Linear 分岐予測器と比較して 4 本鎖構成の Pipelined PTBP で 1.9%，8 本鎖構成で 2.3%の IPC が向上した．また，Idealized PTBP と Pipelined PTBP を比較すると 8 KB 構成では 4 本鎖の Pipelined PTBP で 0.4%，8 本鎖の Pipelined PTBP で 0.1%の IPC 低下であった．パイプライン化による性能の低下は小さいことが確認できる．

図 13 に各ベンチマークでの分岐予測ミス率と IPC を示す．前節の結果と同じく，この結果では従来のパーセプトロンモデルの分岐予測器が苦手としていた 176.gcc の性能を改善している．そのほかの特性も前節とほぼ一致しており，PTBP が現実的な実装で，従来手法と比較して高精度な分岐予測が可能な手法であることが確認できる．

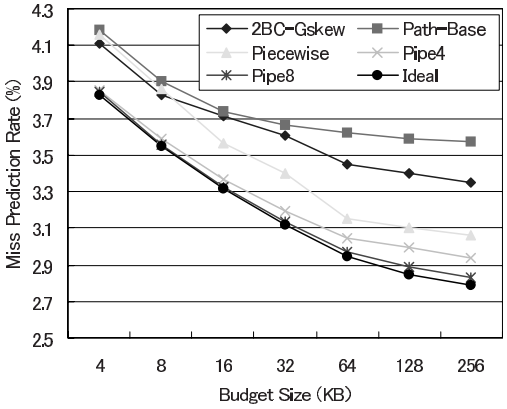

図 11 各分岐予測器での平均の分岐予測ミス率
Fig. 11 Miss prediction rate of each predictor.

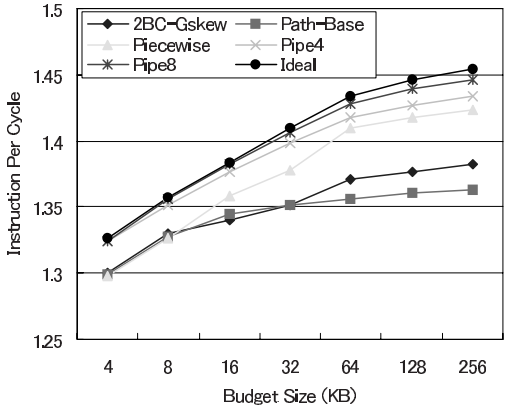

図 12 各分岐予測器での IPC
Fig. 12 IPC of each predictor.

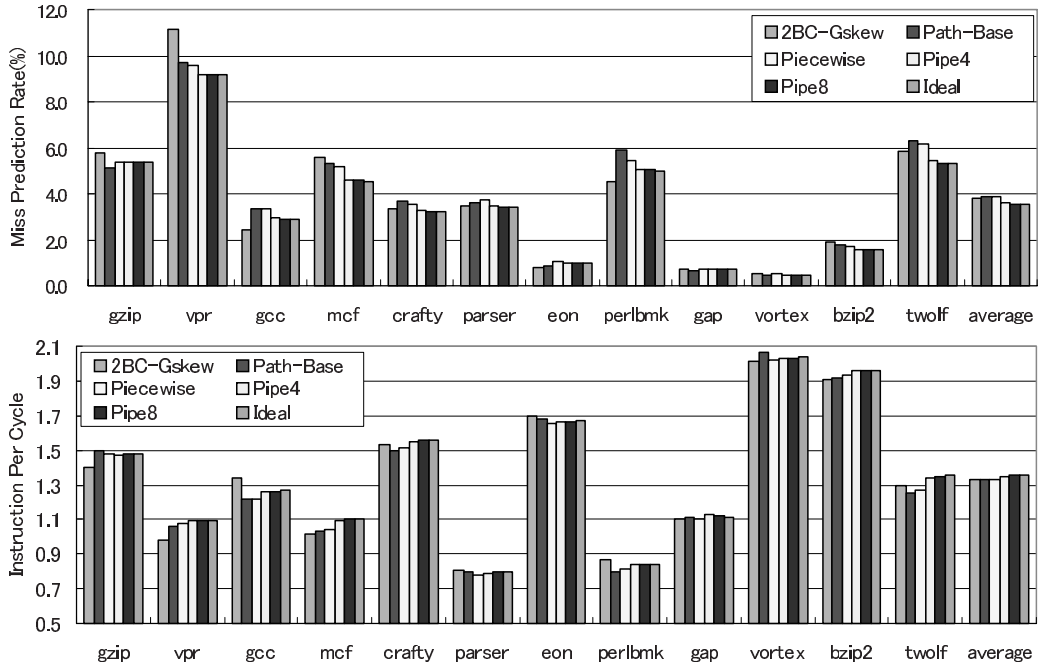

図 13 8 KB 構成での各ベンチマークでの予測精度（上）と IPC（下）
Fig. 13 IPC and miss prediction rate of each benchmark with 8 KB budget.

5. おわりに

本稿では，パス情報を利用した分岐予測手法である Path Trace Branch Prediction を提案した．PTBP はパーセプトロンモデルの分岐予測器で，従来手法と比較して詳細な実行パスの履歴情報を利用することで高い分岐予測精度を実現する．この実行パスの追加情報として過去に実行した分岐命令のアドレスを複数利用した Hash 値を利用した．

また，PTBP を効率的にパイプライン化する方法を示した．パイプライン化した PTBP では分岐予測に必要なレイテンシが 2BC-Gskew 予測器などの 2 ビットカウンタを利用した分岐予測器と同程度になるため，プロセッサ上に実装した際に高い性能を実現することができる．

PTBP をシミュレーションを用いて評価をしたところ，分岐予測器として標準的な 8 KB 構成では 4 本鎖の Pipelined PTBP は従来手法と比較して 8.8%の分岐予測ミス率の削減をできることが分かった．また，パイプラインの深さが 20 段のプロセッサ上でのシミュレーションを行い実行時の IPC を計測した．その結果，従来手法と比較して 8 KB 構成では 4 本鎖構成で 1.9%の IPC の向上が確認された．以上より，分岐予測手法として PTBP を採用することにより，高精度かつ低レイテンシな分岐予測を行うことが可能になり，プロセッサの性能向上に貢献できることを示した．

参考文献

1) Block, H.D.: The perceptron: A model for brain functioning, *Reviews of Modern Physics*, Vol.35, pp.123–135 (1962).
2) Desmet, V., Vandierendonck, H. and De Bosschere, K.: A 2bcgskew predictor fused by a redundant history skewed perceptron predictor, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
3) Gao, H. and Zhou, H.: Adaptive information processing: An effective way to improve perceptron predictor, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
4) Jiménez, D.A.: Fast path-based neural branch prediction, *Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.243–252, IEEE Computer Society (2003).
5) Jiménez, D.A.: Idealized piecewise linear branch prediction, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
6) Jiménez, D.A.: Piecewise linear branch prediction, *ISCA '05: Proc. 32nd annual international symposium on Computer architecture*, pp.382–393 (2005).
7) Jiménez, D.A. and Lin, C.: Dynamic branch prediction with perceptrons, *Proc. 7th International Symposium on High-Performance Computer Architecture* (*HPCA'01*), pp.197–206, IEEE Computer Society (2001).
8) Jiménez, D.A. and Lin, C.: Neural methods for dynamic branch prediction, *ACM Trans. Comput. Syst.*, Vol.20, No.4, pp.369–397 (2002).
9) Lee, C-C., Chen, I-C.K. and Mudge, T.N.: The bi-mode branch predictor, *Proc. 30th annual ACM/IEEE international symposium on Microarchitecture*, pp.4–13, IEEE Computer Society (1997).
10) Loh, G.: The frankenpredictor, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
11) McFarling, S.: Combining Branch Predictors, Technical Report TN-36 (June 1993).
12) Michaud, P.: A ppm-like, tag-based predictor, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
13) Seznec, A.: The o-gehl branch predictor, *The 1st JILP Championship Branch Prediction Competition* (*CBP-1*) (2004).
14) Seznec, A. and Fraboulet, A.: Effective ahead pipelining of instruction block address generation, *ISCA '03: Proc. 30th annual international symposium on Computer architecture*, pp.241–252, New York, NY, ACM Press (2003).
15) Seznec, A. and Michaud, P.: De-aliased hybrid branch predictors, Technical Report RR-3618 (1999).
16) Thomas, R., Franklin, M., Wilkerson, C. and Stark, J.: Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history, *ISCA '03: Proc. 30th annual international symposium on Computer architecture*, pp.314–323, ACM Press (2003).
17) Yeh, T-Y. and Patt, Y.N.: Two-level adaptive training branch prediction, *Proc. 24th annual international symposium on Microarchitecture*, pp.51–61, ACM Press (1991).

(平成 17 年 7 月 29 日受付)
(平成 17 年 11 月 11 日採録)

石井　康雄（正会員）

2004 年東京大学理学部情報科学科卒業．現在，同大学大学院情報理工学系研究科コンピュータ科学専攻修士課程．計算機アーキテクチャに興味を持つ．

平木　　敬（正会員）

1976 年東京大学理学部卒業．1982 年東京大学大学院理学系研究科物理学専門課程退学．1986 年理学博士（東京大学）．1995 年より東京大学教授．この間電子技術総合研究所研究員，IBM 社 T.J. Watson 研究センター客員研究員，東京大学助教授等．計算機・ネットワークの高速化の研究に従事．GRAPE-DR project．Data Reservoir project．