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

2008－ARC－177 (7)
2008－HPC－114 (7)
2008／3／5

パイプラインプロセッサ上でのMIN-TAGE予測器の性能評価

牟田口 公洋† 中田 尚† 中島 康彦†

プロセッサの性能を向上させるため，これまでにさまざまな分岐予測器が提案されてきた．近年はプロセッサのマルチコア化が一般的になっており，小規模で電力効率の高い設計が要求されているが，従来の分岐予測器はこの要求に適合しなくなってきている．そこで小規模・省電力で予測回路の遅延が小さいMIN-TAGE予測器が提案され，容量16KBでミス率0.45%を下回る高い予測精度を示した．しかし，その評価はトレースレベルのシミュレータ上で実行されており，実際のプロセッサ上に実装した場合とでは予測テーブルやレジスタ等の更新のタイミングが異なってくると考えられる．本稿ではMIN-TAGE予測器をパイプラインプロセッサ上に実装し，ベンチマークプログラムで性能を計測することによってどのように挙動が変化するのかについて調べた．その結果パイプラインプロセッサ上に実装すると，MIN-TAGE予測器のインデックスや予測テーブルを更新するタイミングの違いによって，トレースレベルのシミュレータによる結果よりも平均で約0.4%ミス率が上昇し，性能が低下することが分かった．

Performance evaluation of MIN-TAGE branch predictor on processor

Masahiro Mutaguchi,† Takashi Nakada† and Yasuhiko Nakashima†

To improve the performance of processors, various branch predictors have been proposed. Small scale and power-efficient processors are recently required because multi-core processors have been popular. When these condition, MIN-TAGE branch predictor that suits complexity-effective implementation and feasible prediction latency has been proposed. The predictor is said to be able to reduce prediction latency in contrast to the conventional branch predictors and save hardware costs. It is also said the accuracy is high. However, the evaluation of MIN-TAGE predictor has been done based on a trace level simulator. When it is implemented on a real processor, there are differences in timing of updating register and prediction table. This paper shows the detailed implementation of MIN-TAGE predictor on a real processor and its performance using clock accurate simulator. The differences in timing of updating register and predictor table result in performance degradation by an average 0.4 percent as compared to the result of a trace level simulator.

1. はじめに

近年のプロセッサは性能向上のため，複数の命令を順次同時に処理するパイプライン処理や命令レベルの並列性を利用し，同時に複数の命令を実行するスーパースカラ処理を行っている．そのため，パイプライン実行が滞らないようにするために，常に命令を供給することが求められる．このため，分岐予測はプロセッサのパフォーマンス全体に大きな影響を与えるほど重要な技術となってきており，高精度な分岐予測器が常に求められている．

これまでにも多くの分岐予測手法が提案されているが，従来手法の多くは大規模なコア上に実装することが想定されており，十分なメモリ容量と大きな予測遅延を許容していた．しかし，近年は1つのLSI上に複数のプロセッサコアを搭載するマルチコア化が大きな流れになっており，そのため小規模でかつ電力効率の高い予測器が求められている．しかし，従来の分岐予測器はこれらの要求に適合しなくなってきており，そこで石井らによって小規模，省電力コア向けに小資源での分岐予測手法としてMIN-TAGE分岐予測器[2]が提案された．このMIN-TAGE分岐予測器は高い予測精度だが複雑で予測遅延の大きいTAGE分岐予測器[4]を元に効率的にパイプライン化し，TAGE分岐予測器のクリティカルパスであった予測テーブルのインデックス用ハッシュ値計算，予測テーブルアクセス，および予測結果の生成という一連の動作を3サイクルにわけて行うことで予測遅延を小さくしている．予測精度も非常に高く，メモリの容量が1KBから16KB全てにおいて既存のプロセッサで広く採用されているgshare予測器はもちろんハイブリッド型の予測器で

† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

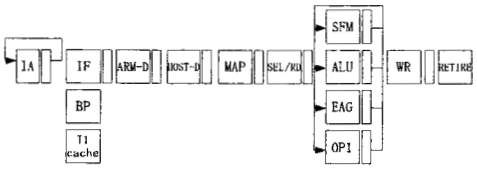

図 1 プロセッサのパイプライン構成

ある BIMODE 予測器[5] や 2BCGSK 予測器,[3] また Championship Branch Prediction において高い評価を得た TAGE 予測器や GEHL 予測器[6] を上回る精度を達成している．ところが，これらは実際のプロセッサ上で行われた評価ではなくトレースレベルのシミュレータ上で評価されている．実際のプロセッサではトレースシミュレータとは挙動が異なることから，分岐予測器の性能にも影響してくると考えられる．本研究では，MIN-TAGE 予測器をパイプラインプロセッサ上に実装し，トレースレベルのシミュレータと比較したときどのように性能が変わってくるのかベンチマークプログラムを使用し，ミス率を計測することによって評価する．以降，2 章では MIN-TAGE 予測器を実装するプロセッサのパイプライン構成について説明し，3 章で実装した方法について述べる．4 章ではプロセッサに MIN-TAGE 予測器を実装した際に生じるトレースシミュレータとの違いを述べ，5 章ではプロセッサ上で評価した結果と考察について述べる．6 章ではまとめを述べる．

2. プロセッサのパイプライン構成

本章では，MIN-TAGE 予測器を実装するプロセッサのパイプライン構成について述べる．

本研究では ARM のスーパスカラプロセッサを使用する．パイプライン構成を**図 1** に示す．

IA ステージではフェッチする命令のアドレスを生成する．IF ステージでは命令キャッシュ（I1-cache）から連続 2 命令を読み出すと同時に，gshare 分岐予測器（BP）を用いて分岐予測を行う．命令デコーダ（ARM-D）では ARM 命令を RISC 型内部命令に変換する．また，このステージで分岐命令が判明し，予測が Taken の場合，分岐先にジャンプする．命令デコーダ（HOST-D）では実行条件付き命令を条件に関わらず依存関係が変化しない命令列に分解する．MAP ステージ論理レジスタから物理レジスタへリネーミングを行う．SEL/RD ステージ演算器からのバイパスが利用可能かどうか調べるとともに，依存関係の待ち合わせと命令発行を行う．SFM でシフト演算と積和演算用の補助演算，ALU は一般的な加減算や論理演算，EAG はアドレス計算と積和補助演算の選択，OP1 はデータキャッシュおよびストアバッファを対応付ける．WR ステージでは物理レジスタに演算結果の書き込みを行う．RETIRE ステージ先行命令が全て完了した命令を物理レジスタから論理レジスタに変える．

3. 実　　装

実装した MIN-TAGE 予測器の全体構成を**図 2** に示す．MIN-TAGE 予測器は 4 つのテーブルと Enhanced Folded Index レジスタ，タグ比較回路，セレクタで構成されている．T0 から T3 の 4 つのテーブルのうち、T0 は BIM と呼び，分岐命令の PC をインデックスに持つ BiMode 予測器である。また、T1 から T3 はそれぞれ Enhanced Folded Index の値をインデックスに持つ予測テーブルである。これらのテーブルはそれぞれ分岐予測に用いる飽和カウンタ (CTR) と有効性を示す飽和カウンタ (U) を持ち、T1 から T3 はタグ付きであり、タグが一致しなかった場合はその予測値は用いられない。T1 から T3 すべてのテーブルでタグが一致しなかった場合は BIM の値が使われる．また Enhanced Folded Index は過去のパス情報と分岐履歴を保存したレジスタ[7] であり，その構成方法を**図 3** に示す．この図は R0-7 の 8bit の Enhanced Folded Index の例である．GHR と PHR はそれぞれ分岐履歴とパス情報を保持するシフトレジスタであり，H0，P0 はそれぞれ直前の分岐命令の分岐結果と，その分岐命令の PC の一部分である．1 つの分岐が実行される毎にこれらの情報は更新される．Enhanced Folded Index は単純なシフトレジスタである GHR や PHR と異なり，図に示すように循環的に XOR を取ることにより，少ないビット数で長期間の分岐履歴情報を含むことができる．MIN-TAGE 予測器ではこれらのレジスタを分岐予測テーブルの参照に用いる．

MIN-TAGE 予測器は予測値を生成するため，まずは予測テーブルから行選択によって予測値の候補となる Row Data を読み出し，次の列選択のサイクルで予測値を生成するエントリを行選択で選んだ候補の中から 1 つに決定する．そして，次のサイクルで予測結果を出力する．このように，予測値を生成するのに 3 サイクルを要するので，IA ステージで PHT の行選択，IF ステージで列選択，そして ARM-D ステージで予測値を生成するように実装する．予測部分の詳細な動きを，図 3 の例を使用して説明する．

- 予測 1 サイクル目
  IA ステージで Enhanced Folded Index レジスタの中で次のサイクルでも変化しない値，すなわち R0,2,3,4,5 を使用して PHT の行選択をし，読み出したデータをパイプラインレジスタに格納する．
- 予測 2 サイクル目
  IF ステージでは Enhanced Folded Index レジスタの中で 1 サイクル目から変化して正しい値が確定した R0,2,7 を使用して PHT の列選択をし，エントリを決定する．このとき，同時に PC を使

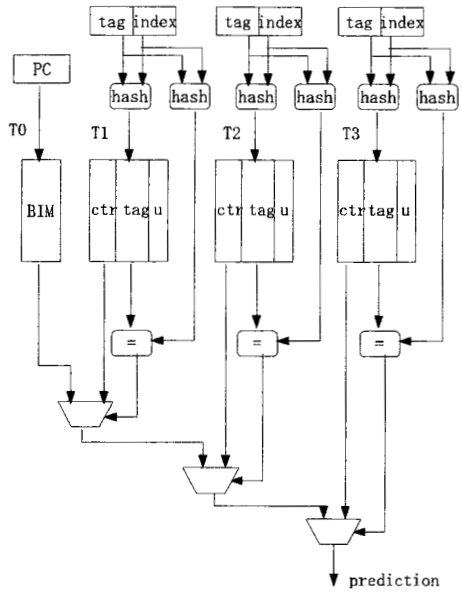

図 2 実装した MIN-TAGE 予測器の全体構成

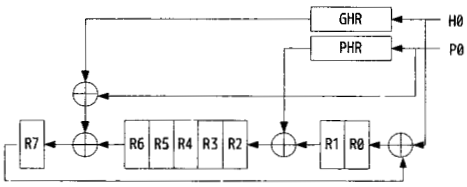

図 3 Enhanced Folded Index

用して BIM の行選択をしパイプラインレジスタに格納する.

- 3 サイクル目
  ARM-D ステージでは PC を使用して BIM の列選択をし予測値を決定するとともに分岐アドレスと Enhanced Foleded Index から生成されたタグと選択したエントリのタグとを比較して，タグが一致し履歴長が最も長いエントリから予測値を生成する．もしタグが一致するエントリが存在しない場合は BIM の予測値を最終的な予測結果とする.

上記では Enhanced Folded Index レジスタが 8 ビットの場合の例を示したが，10 ビット使用するときは行選択に 6 ビット，列選択に 4 ビット使用するようにした.

例えば，GHR を 13 ビットで，PHR が 1 ビットのときは図 4(a) の R0,R3,R4,R5 が直前のサイクルで更新されるのでこれらの bit を列選択に使用し，行

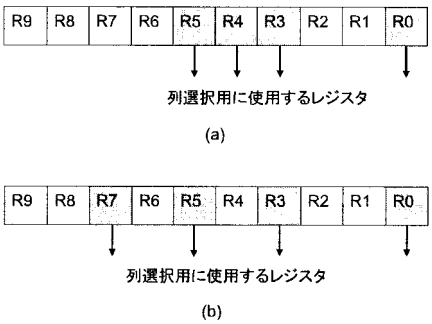

図 4 phr1 ビット，2 ビット使用時の行・列選択インデックス

選択には残りの R1,R2,R6,R7,R8,R9 の値を 1 つ前のサイクルで使用する．同様に PHR が 2 ビットのときは図 4(b) の R0,R3,R5,R7 を列選択に使用し，R1,R2,R4,R6,R8,R9 を行選択に使用する.

次に予測テーブルの更新について述べる．更新は分岐命令の結果が判明し，実行が完了する RETIRE ステージで行う．予測がミスした場合，予測値を生成したテーブルが T3 以外のとき，予測テーブルに新しいエントリを追加する.

- 予測値を生成したテーブルより長い履歴長を使用しているテーブルの中で，有効性を示すカウンタである U の値が 0 のエントリを探す.
- もしあればそのエントリに予測に使用したタグを代入し，結果が Taken だったときは 0，Not Taken だったときは-1 を CTR に代入する．U は 0 を代入する.
- U の値が 0 のエントリが存在しなかったとき，予測値を出したテーブルよりも長い履歴長を使用する PHT の中からランダムにテーブルを選択し，予測に使用したタグで置き換える.
- 同時に予測値を生成したテーブルのエントリの CTR を Taken のときインクリメント，Not Taken のときデクリメントする.
- 予測値を生成したテーブルが PHT の場合 U もデクリメントして更新する.

予測がヒットした場合，エントリの置き換えはせず，予測値を生成したテーブルのカウンタ値を更新し，有効性を示すカウンタである U の値をインクリメントする.

次に，GHR，PHR および Enhanced Folded Index の更新について述べる．まず，GHR は ARM-D ステージで予測値を使用し，投機的に更新することとする．したがって，予測値と実際の分岐結果が異なっているとき，これらのレジスタは間違った値を含んでいることになる．そこで RETIRE ステージで分岐予測ミスを検出したときにこれらのレジスタを補正することとする．PHR についても分岐予測した方向の命令列を投機実行しているので，実際には実行されない

パスの情報が含まれており，GHR 同様補正の必要がある．そして Enhanced Folded Index も同様に，投機的に更新していた GHR と PHR で構成されているので補正する必要がある．上記レジスタを補正するためにはこれらのレジスタをもう一組用意する．一方は予測用に ARM-D ステージで投機的に更新し，もう一方は RETIRE ステージで分岐命令の結果が判明したのちに更新する．そして，予測値が間違いとわかったとき，すなわち分岐予測ミスが発生した場合は，その時点で ARM-D ステージで投機的に更新したレジスタの値は不正であるので，RETIRE ステージで更新していた正しいレジスタの値を用いて修正する．これで次の分岐命令から正しい情報を使用して予測することができる．

4. プロセッサとトレースレベルシミュレータの相違点

本研究で使用したプロセッサに MIN-TAGE 予測器を実装した際に生じる，トレースシミュレータとの違いについて述べる．

プロセッサとトレースシミュレータの大きな違いとして予測テーブルやインデックスに使用するレジスタの更新のタイミングがあげられる．パイプラインプロセッサは，複数のパイプラインステージによって構成されており，パイプラインの前方のステージで予測値を生成して，予測方向へとジャンプする．

インデックスの値は前述のように予測結果が確定する ARM-D ステージにおいて，投機的に更新することができるが，投機的に更新されたインデックスを利用した後続の分岐命令の予測結果にはトレースシミュレータによる結果と相違が生まれる可能性がある．

また，MIN-TAGE 予測器では予測ミスが発生した場合に新たなエントリを予測テーブルに作成するが，予測ミスが確定するのは分岐命令の実行が完了するパイプラインの後方のステージであり，これを投機的に行うことは不可能である．つまり，予測からテーブル更新までパイプラインのステージ数分だけサイクル数が必要となるため，実行される分岐命令の間隔が小さいと直前のいくつかの分岐の結果が，予測テーブルやレジスタに反映されないまま次の予測をすることになる．

これは前述のようにまた，1 クロックで複数の命令をフェッチまたはデコードできるので，厳密に 1 分岐命令毎にレジスタを更新することは不可能である．したがって，1 クロックで複数の分岐予測を行う場合には必然的に同じインデックスを用いることになる．このようにパイプラインプロセッサでは実行される分岐命令の間隔により予測結果に反映されている分岐命令の範囲が変化する．一方，MIN-TAGE 予測器をトレースシミュレータに実装した場合**図 5** のようなタイミングチャートになる．例えば I=0 が予測するとき，T=0 で PHT の行選択を行い，T=1 で列選択を行う．そして T=2 で予測結果を出力する．このとき T=1 で列選択をするが，これは列選択用のインデックスとして I=-2 が T=0 で予測した結果を Enhanced Folded Index にフォワーディングし，反映されたものを使用する．つまりトレースシミュレータの場合，常に 2 個前の正しい分岐結果がインデックスに反映された状態にある．

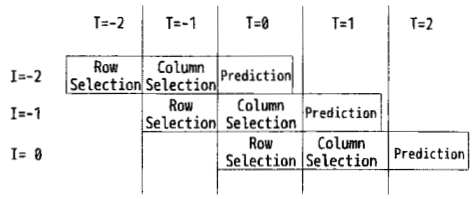

図 5 MIN-TAGE 予測器のタイミングチャート

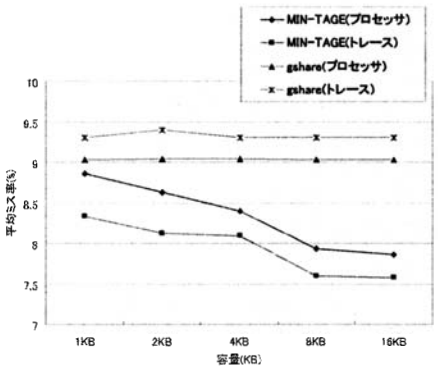

図 6 gshare 予測器，MIN-TAGE 予測器の予測ミス率

5. 評価と考察

評価には Stanford Benchmark を用い，プログラム 10 個の平均ミス率を測定した．測定した結果を**図 6** に示す．結果は gshare 予測器と MIN-TAGE 予測器をそれぞれ，トレースシミュレータ上で測定した場合とパイプラインプロセッサ上で測定した場合を示す．図 6 より MIN-TAGE(トレース)，MIN-TAGE(プロセッサ)，gshare(プロセッサ)，gshare(トレース) の順に低いミス率となった．また，gshare 予測器はプロセッサ上，トレースシミュレータ上ともに 1KB から 16KB の容量でミス率の変化がほぼ見られなかったのに対し，MIN-TAGE 予測器は容量を増やすとどちらもミス率が減少した．また，プロセッサ上の MIN-TAGE 予測器はトレースシミュレータ上よりもミス率が高くなり，最大で 0.52%のミス率上昇となった．

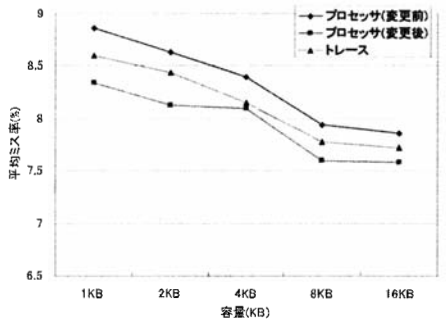

図 7 インデックスの選択方法変更後の予測ミス率

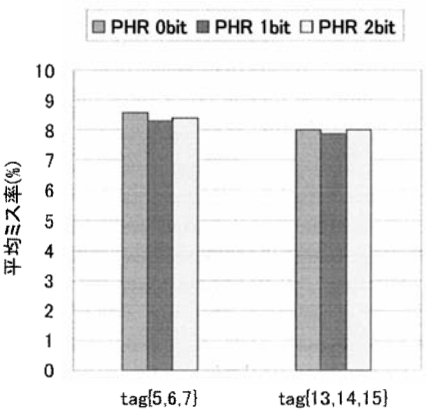

図 8 PHR による影響

MIN-TAGE 予測器について，パイプラインプロセッサ上での予測ミス率低下の要因は，前章で述べたパイプラインプロセッサとトレースシミュレータ上における予測器の挙動の違いからくるといえるが，その一つが Enhanced Folded Index が更新されるタイミングにあると考えられる．まずトレースの場合，毎サイクル予測結果が Enhanced Folded Index にフォワーディングされシフトされるので行選択と列選択で Enhanced Folded Index は異なっていた．ところがプロセッサに組み込んだ場合，行選択をする IA ステージと列選択をする IF ステージの間でタイミングよく予測結果がフォワーディングされない限り，行選択と列選択で Enhanced Folded Index は同じ値になる．そうなると，例えば図 3 の例でいうと，列選択で使用する R0, R2 が行選択で使用した R0, R2 と同じになってしまい，テーブルのアクセス範囲が狭まってしまう．このことによる影響を調べるため，IF ステージで列選択するときに IA ステージと Enhanced Folded Index が同じだったとき，結果のフォワーディングによるシフトがなかったとみなし，列選択のインデックスに行選択で使用されなかったレジスタを使用するようにして，ミス率を計測した．その結果の予測ミス率を**図 7** に示す．結果，全ての容量においてミス率が下がった．トレースとのミス率が差はもともと平均約 0.4%あったが，上記のように変更すると平均約 0.2%に縮まった．

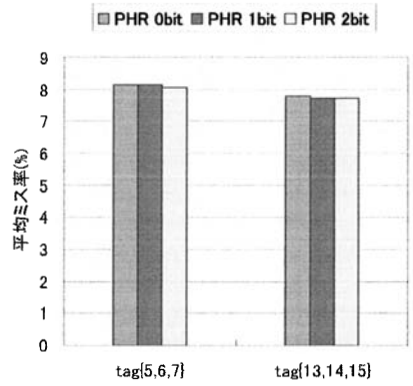

図 9 インデックスの選択方法変更後の PHR の影響

5.1 PHR の影響

また，行選択と列選択で同じ Enhanced Folded Index レジスタを使用することは Enhanced Folded Index レジスタに使用する PHR の長さによって予測ミス率に影響してくる．PHR を 0 から 2 ビットに設定し，T1 から T3 の各テーブルのタグ長が 5,6,7 の場合と 13,14,15 の場合についてプロセッサ上で MIN-TAGE 予測器の予測ミス率を計ると**図 8** に示す結果となった．この結果から PHR を 1 ビット使用したときが 1 番ミス率が低くなった．2 ビット使用するとミス率が上昇し，タグ長が 13,14,15 の場合だと使用しないときとほぼ同じになった．その理由として，上記と同じで，行選択と列選択で Enhanced Folded Index 同じ値だったことが原因と考えられる．例えば PHR が 1 ビットについて，図 4 の例でみると行選択と列選択で Enhanced Folded Index が同じだと列選択の R0, R5 の 2 ビットが行選択のものと同じになる．一方 PHR が 2 ビットの場合，列選択の R0,R3,R5,R7 全てのビットが行選択のものと同じ値のものとなる．そのため，行選択のインデックスが決まると列選択のインデックスが一意に決まってしまう．その結果，同じ行選択のインデックスを持つアドレス同士で競合が発生したと考えられる．そこで，再びこのことを考慮して，列選択で使用するインデックスのレジスタを選択した場合の予測ミス率を**図 9** に示す．その結果，PHR を 2 ビット使用したときに最もミス率が低くなった．

5.2 カウンタのビット数の影響

次に PHT を構成するカウンタのビット数による影響について調べる．まず，カウンタ CTR のビット数の影響を調べるため容量 1KB, 2KB, 4KB, 8KB, 16KB

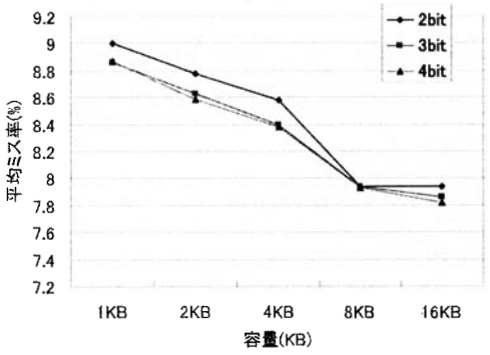

図 10 CTR のビット数とミス率

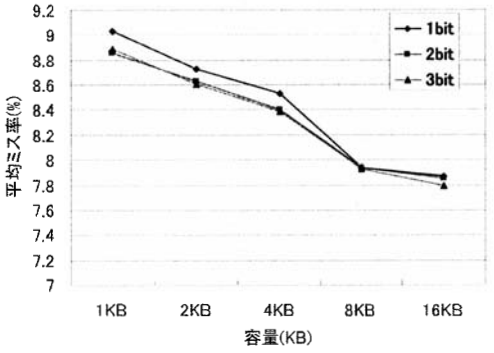

図 11 U のビット数とミス率

構成についてカウンタ U を 2 ビットに固定し，カウンタ CTR を 2，3，4 ビットと設定したときの平均ミス率を調べた．結果を**図 10** に示す．

結果はカウンタ CTR のビット数が 4 ビット，3 ビット，2 ビットの順にミス率が低くなった．また，2 ビットを使用するよりも 3 ビットや 4 ビットを使用したほうが 1KB，2KB，4KB 構成でミス率が 0.2%と相対的に大きく減少した．それに対し 3 ビットと 4 ビットを比較した場合，4 ビットの方がわずかに低いミス率を示すがほぼ同じ結果となった．

つづいてカウンタ U のビット数による影響を調べる．上記と同じく容量 1KB から 16KB の構成についてカウンタ CTR を 3 ビットに固定し，カウンタ U を 1，2，3 ビットと設定したときの平均ミス率を調べた．結果を**図 11** に示す．

結果は容量が 1KB，2KB，4KB の構成のとき，カウンタ U のビット数を 1 ビット使用したときと 2 ビット，3 ビット使用したときの平均ミス率は相対的に大きな差があり，2 ビット，3 ビット使用した方が最大で 0.17%低いミス率となった．8KB，16KB 構成の場合については全てほぼ同じミス率となった．また，2 ビットと 3 ビット使用したときについては全体的にほぼ同じミス率で最高でも 0.04%の差しかなかった．

6. おわりに

本報告では，MIN-TAGE 予測器をパイプラインプロセッサ上とトレースシミュレータ上に実装した場合の挙動の違いについて述べた．そしてその違いによって MIN-TAGE 予測器は，プロセッサ上ではトレースシミュレータ上よりもミス率が平均約 0.4%増加した．そしてその増加原因の一つが行選択と列選択の時点で Enhanced Folded Index が常に 1bit シフトしているという仮定がくずれ，これらの値に同一の bit を参照してしまうために生じたことが分かった．この問題を解消するために，Enhanced Folded Index の全てのレジスタを正しく使用するようにして，予測テーブルのエントリの選択をした結果ミス率の増加が平均約 0.2%まで削減した．

参考文献

1) Scott McFarling, “Combining branch predictors”, Digital Western Research Laboratory Technical Note TN-36 (1993)
2) 石井 康雄 , 平木 敬, “小規模・省電力コアのための省資源分岐予測方式”, 情処研報 2007-ARC-174, p.205-210 (2007)
3) André Seznec ,Pierre Michaud, “De-aliased Hybrid Branch Predictors”, Technical report, INRIA RR-3618 (1999)
4) André Seznec ,Pierre Michaud, “A case for (partially)-tagged geometric history length predictors”, Journal of InstructionLevel Parallelism (2006)
5) Kenji Kise, Takahiro Katagiri, Hiroki Honda and Toshitsugu Yuba,“The bimode++ branch predictor”, Proceedings of the Innovative Architecture on Future Generation Hight-Performance Processors and Systems, p.19-26 (2005)
6) André Seznec, “Analysis of the o-gehl branch predictor”, Proceedings of the 32nd Annual International Symposium on Computer Architecture (2005)
7) Yasuo Ishii, “Fused two-level branch prediction with ahead calculation”, In The 2nd JILP Championship Branch Prediction Competition(CBP2), (2006)