社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コンテキスト・ベース値予測を利用した分岐先予測器

平嶋　哲朗† 嶋田 創†† 三輪 忍††† 富田　眞治††

† 株式会社野村総合研究所
†† 京都大学大学院情報学研究科　〒 606–8357 京都市左京区吉田本町
†† 東京農工大学工学府　〒 184–8588 東京都小金井市中町 2-24-16
E-mail: †{hirasima,shimada,tomita}@lab3.kuis.kyoto-u.ac.jp, ††smiwa@cc.tuat.ac.jp

あらまし　命令レベル並列性の抽出を阻害する要因の 1 つとして，分岐命令に起因する制御依存がある．この制御依存による制限を緩和するため，現在のプロセッサでは分岐方向予測や分岐先予測が用いられている．この分岐先予測として主に使われているものに分岐先バッファ(BTB) がある．しかし，BTB では複数の分岐先を取りうるレジスタ間接分岐に対する予測成功率が低いという問題がある．本論文では，複数の分岐先予測を行う方法の 1 つとして，分岐命令のローカルの分岐先履歴をもとに分岐先予測を行う手法を提案する．提案する予測器による IPC の向上を評価した結果，BTB も含めて 40KB のコストの履歴長 4 で 2.6%向上し，96KB の履歴長 5 の場合は 4.0%向上するという結果を得た．

Branch Target Predictor Utilizing Context Base Value Predictor

Tetsuro HIRASHIMA†, Hajime SHIMADA††, Shinobu MIWA†††, and Shinji TOMITA††

† Nomura Research Institute, Ltd.
†† Graduate School of Engineering, Kyoto University
†† School of Engineering, Tokyo University of Agriculture and Technology
E-mail: †{hirasima,shimada,tomita}@lab3.kuis.kyoto-u.ac.jp, ††smiwa@cc.tuat.ac.jp

Abstract　A control dependency is one of the factor which limits instruction level parallelism. To alleviate limitation from the control dependency, current processors use techniques called branch target prediction and branch direction prediction. Current major processors uses a Branch Target Buffer or BTB to predict branch target. But BTB cannot suit for a register indirect branch which has several branch targets. In this paper, we propose the branch target predictor which is based on a lobal branch target history of the branch instruction. Our evaluation result shows that the mechanism under 4 history length which is achieved with 40KB hardware cost (including BTB) achieves 2.6% IPC improvement. Also, the mechanism under 5 history length (96KB) achieves 4.0% IPC improvement.

1. はじめに

近年の高性能プロセッサの多くは，out-of-order 実行やスーパスカラ実行などの命令レベル並列性 (ILP: Instruction Level Parallelism) を抽出する技術を利用し，高い性能を達成している．この ILP の抽出を阻害する要因の 1 つとして，分岐命令に起因する制御依存がある．この制御依存の問題を緩和するため，従来より，分岐予測を用いた投機実行という手法が使われて来た．これは，分岐方向と分岐先を予測して命令をフェッチし，それらの命令を投機実行するものである．この分岐予測という手法は，大きく分けて，分岐方向予測と分岐先予測という 2 つの手法に分かれている．分岐方向予測とは，条件分岐が成立するかしないかを予測する手法であり，2bc や gshare などの方式がある．分岐先予測とは，分岐先の命令アドレスを予測する方式であり，分岐先バッファ(BTB: Branch Target Buffer) などの方式がある．分岐予測による投機実行は，以下のように行われる．まず，分岐方向予測で分岐の成立／不成立を予測する．分岐が成立すると予測された場合 (もしくは無条件分岐の場合)，分岐先予測によって分岐先の命令アドレスを予測し，そこから命令フェッチを行う．分岐が不成立と予測された場合，分岐命令の次の命令アドレスから命令フェッチを行う．

分岐先予測として一般的に使われる BTB は，その分岐命令

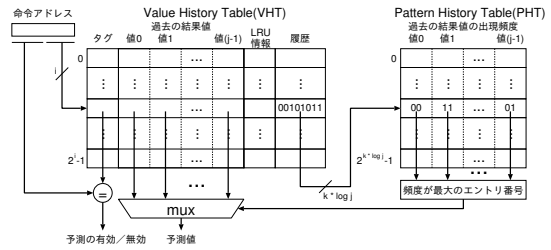

図 1　コンテキスト・ベース値予測器 (予測時の動作)

が最後に実行された時の分岐先アドレスを保持して予測アドレスとするため，複数の分岐先を取りうる間接分岐では効果が薄くなる．近年増加しているオブジェクト指向言語では、複数の分岐先を持つレジスタ間接分岐が多用されるため、間接分岐の複数の分岐先を予測する研究が数多く行われている．

本論文では，間接分岐の複数の分岐先の予測を行う方法の 1 つとして，分岐命令のローカルの分岐先履歴をもとに分岐先予測を行う手法を提案する．この分岐先予測器の構成は，命令の実行結果のローカル履歴を利用するという点で同じである，コンテキスト・ベース値予測器の構成をベースとする．先行研究において，グローバル分岐方向履歴と関連させて分岐先を予測する手法は多数提案されているが，我々の提案のように，ローカルの分岐先履歴を用いるものは存在しない．提案する予測器による IPC(Instructions Per Cycles) の向上を評価した結果，BTB も含めて 40KB というハードウェア・コストの履歴長 4 で 2.6%向上し，96KB というコストの履歴長 5 の場合は 4.0%向上するという結果を得た．

2. 関 連 研 究

この節では，関連研究として間接分岐のための分岐先予測の先行研究と，提案する分岐先予測器のベースとなるコンテキスト・ベース値予測について述べる．

2.1 レジスタ間接分岐命令のための分岐先予測

間接分岐の複数の分岐先を予測しようという試みは数多く行われている．この中の多くの研究においては，予測対象となる間接分岐に至るまでの制御フローによって分岐先が変化するものと考え，グローバル分岐方向履歴によって分岐先を選択するというアプローチを取っている．Target Cache [1]，Cascaded Predictor [2]，VPC Predictor [3] などは，いずれもグローバルな分岐履歴を用いており，おおまかに言えば，これらの研究間の差異はグローバルな分岐履歴の用い方や予測表の構成方法にある．

上記の方法とは全く異なるアプローチを取るものとして，レジスタ間接分岐ターゲット・フォワーディング [4] がある．これは，仮想関数による分岐の分岐先アドレスはメモリから読み出さなくてはならないことに着目し，メモリから読み出した分岐先アドレスを，レジスタを介さずにフロントエンドに送る手法である．この手法は予測ではないため，他の間接分岐の分岐先予測手法と組み合わせることが可能という利点もある．

2.2 コンテキスト・ベース値予測

プログラム中からさらなる ILP を抽出するための手法の 1 つとして，データ依存による制約を緩和する，値予測という手法がある．この値予測の手法としては，最終値予測 [5]，ストライド予測 [6]，コンテキスト・ベース予測 [7] などがあげられる．

これらの手法の中で，コンテキスト・ベース予測は本論文で提案する手法のベースとなっているため，その概要を説明する．コンテキスト・ベース予測は，結果値の出現パターンに規則性がある場合，その過去の出現履歴を用いて，次の結果値を予測する手法である．以下，例をあげて説明する．ある命令の結果値が a,b,c の 3 種類を取り，“a a b c a a b c ...” というパターンで出現するものとする．前記のパターンの場合，過去 4 回の履歴が分かっていれば，次に出現するパターンを一意に決めることができる．例えば，過去 4 回の履歴が abca ならば次に出現する値は a であり，caab ならば次の出現する値は c である．

この値の履歴とそれに応じた予測値を出力するため，図 1 に示されるような構成のコンテキスト・ベース値予測器が提案されている．予測器は大きく分けて **Value History Table(VHT)** と **Pattern History Table(PHT)** から構成される．VHT は値予測の対象となる命令の命令アドレスの下位でインデクスされ，タグ，複数の過去の結果値，過去の結果値の置き換えのための LRU 情報，値の出現履歴を持つ．値の出現履歴は，過去の結果値が登録されているエントリ番号を使って表される．例えば，VHT の 1 エントリあたり，4 個の過去の結果値を保持でき，エントリ 0 に a という値，エントリ 1 に b という値，エントリ 2 に c という値が保持されている場合，aabc という履歴は 00000110 という 2 進数で表される．PHT は VHT から読み出された履歴でインデクスされ，当該履歴が現れた後の，各エントリの結果値の出現頻度を示す，出現頻度カウンタを保持する．出現頻度カウンタは，一般的に $n$bit 飽和型カウンタで構成され，更新時には，出現した値に対応するカウンタを増加させ，それ以外のカウンタを減少させるという操作を行う．なお、予測時の動作と VHT と PHT の更新動作の詳細については、3.1 節の説明と重複するため、ここでは省略する。

この値予測を分岐方向予測に利用した関連研究として，文献 [8] がある．これは，条件分岐の条件判定に必要な値を値予測で高い精度で予測できる場合，値予測器で得られた結果によって分岐方向を予測するというものである．

3. コンテキスト・ベース分岐先予測器 (CB-BTP) の提案

本論文では，コンテキスト・ベース値予測器を利用して，各間接分岐のローカル分岐先履歴を利用した分岐先予測を行う，**コンテキスト・ベース分岐先予測器 (CB-BTP: Context-Based Branch Target Predictor)** を提案する．これは，用いる履歴が異なるという点で，2.1 節に示した，間接分岐命令に至るまでのグローバル分岐履歴やグローバルな間接分岐の分岐先 (の下位ビット) の履歴を用いる方法と異なる．グローバル分岐履歴とローカル分岐履歴は分岐方向予測でも用いられており、また、さらなる予測成功率の向上を目指したそれらのハイブリッド構成も存在する。分岐先予測においても、過去のグローバル履歴を利用する提案とローカル履歴を利用する本提案のハイブ

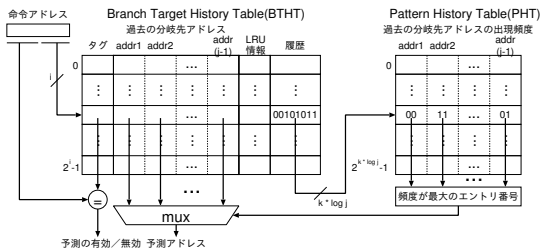

図 2　コンテキスト・ベース分岐先予測器 (CB-BTP)

リッド構成によって、さらなる予測成功率の向上が期待できる。

3.1　CB-BTP の構成と動作

図 2 に示すように，提案する CB-BTP の構成と動作は，コンテキスト・ベース値予測器のそれとほぼ同じである．図 1 からの変更点は以下の通りである．

- VHT が **Branch Target History Table(BTHT)** となる．
- VHT に登録する命令が分岐命令となる．
- 過去の結果値の代わりに過去の分岐先アドレスが登録される．

以下，予測時の動作と BTHT と PHT の更新動作について説明する．予測時には，以下のような動作を行う．

（1） 分岐命令の命令アドレスの下位ビットで BTHT を引き，タグ，過去の分岐先アドレス，履歴を読み出す．

（2） 分岐命令の命令アドレスの上位ビットとタグを比較し，過去に BTHT に分岐先アドレスを登録した分岐命令と同じ命令かどうかを確認する．同時に，読み出された履歴で PHT を引き，各分岐先アドレスの出現頻度を読み出す．

（3） PHT より読み出した各分岐先アドレスの出現頻度を比較し，出現頻度が最大のものを決定する．

（4） (3) の出力によってマルチプレクサを駆動し，出現頻度が最大の過去の分岐先アドレスを予測アドレスとする．

分岐命令の実行後の表の更新は，以下のような手順で行う．

（1） 分岐命令アドレスの下位ビットで BTHT を引き，タグ，過去の分岐先アドレス，履歴，LRU 情報を読み出す．

（2） タグの比較を行う．タグが一致しない場合，BTHT のエントリを初期化し，生成された分岐先アドレスの登録を行い，履歴，LRU 情報の初期化を行う．その後，初期化された履歴で PHT を更新し，更新動作を終了する．

（3） (2) でタグが一致した場合，生成された分岐先アドレスと過去の分岐先アドレスを比較する．過去の分岐先アドレスと一致しない場合，LRU 情報によって，最も出現頻度が低い過去の分岐先アドレスと入れ換えを行い，そのエントリ番号を使って履歴，LRU 情報を更新する．その後，更新された履歴で PHT を更新し，更新動作を終了する．

（4） (3) で生成された分岐アドレスが過去の分岐先アドレスと一致した場合，そのエントリ番号を使って履歴と LRU 情報を更新する．その後，更新された履歴で PHT を更新し，更新動作を終了する．

なお，本論文では，PHT 中の各分岐先アドレスに対応する出現頻度カウンタは 2bit 飽和型カウンタで構成され，更新時は，出現した値については+1 を，それ以外の値については −1 を加算することとした．

表 1　プロセッサの構成

プロセッサ・コア	発行幅 8, RUU 128 エントリ, LSQ 64 エントリ, int ALU 8, int mult/div 4, fp ALU 8, fp mult/div 4, メモリ・ポート 8
分岐予測	PHT 32K エントリ／履歴長 11 の gshare, RAS 16 エントリ、ミス・ペナルティ 15 サイクル
L1 I-キャッシュ L1 D-キャッシュ	64KB/32B ライン/2-way ヒット・レイテンシ 2 サイクル
L2 統合キャッシュ	2MB/64B ライン/4-way ヒット・レイテンシ 16 サイクル
メモリ	初期参照 128 サイクル, 転送間隔 2 サイクル
TLB	命令 16 エントリ, データ 32 エントリ ミス・レイテンシ 144 サイクル

表 2　ベンチマークごとの間接分岐出現率とその分岐先予測成功率

benchmark	間接分岐 / 全分岐	リターン以外の間接分岐 / 全分岐	予測成功率
deltablue	43.10%	10.31%	99.37%
richards	39.42%	6.84%	85.24%
bzip2	2.17%	0.01%	78.01%
eon	23.87%	5.54%	70.63%
gcc	8.06%	2.35%	37.11%
gzip	5.19%	0.01%	99.95%
mcf	6.96%	0.03%	100.00%
parser	12.40%	0.01%	92.42%
perlbmk	21.33%	8.29%	36.18%
vortex	12.53%	0.43%	46.96%
vpr	7.32%	0.12%	77.37%
aster	19.61%	1.55%	100.00%
bzip2	0.77%	0.01%	56.62%
h264ref	9.42%	2.37%	99.17%
mcf	5.61%	0.07%	99.10%
libquantum	0.59%	0.01%	87.65%
omnetpp	0.01%	0.01%	59.38%
sjeng	7.84%	3.32%	44.36%
dealII	0.01%	0.01%	65.74%
namd	6.85%	0.45%	91.63%
soplex	0.01%	0.01%	59.38%

3.2　CB-BTP と BTB のハイブリッド構成

3.3 節で詳細を説明するように，CB-BTP は非常にハードウェア・コストが大きい．そのため，間接分岐以外の分岐や動的には分岐先が 1 つしかない間接分岐などの，BTB でも正しく分岐先を予測可能な分岐に CB-BTP のエントリを割り当てるのは好ましくない．そのため，CB-BTP と BTB の両方を準備し，実行当初は BTB を用い，複数の分岐先が出現した場合にのみ CB-BTP を利用するハイブリッド構成を取ることが望ましい．

図 3 に，CB-BTP と BTB のハイブリッド構成を示す．BTB の各エントリには，同エントリに保持されていた分岐の予測が CB-BTP に移管されたことを示すビット (CB-BTP 移管ビッ

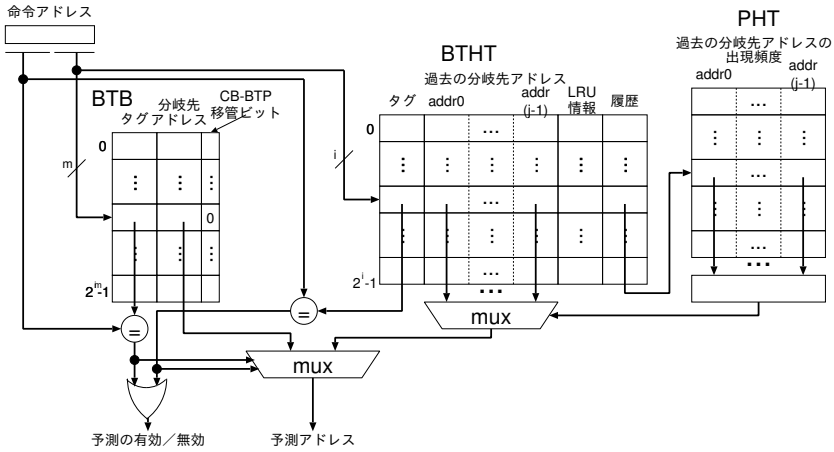

図 3　BTB と CB-BTP のハイブリッド構成

表 3　n 番目の出現頻度の分岐先に分岐する確率

	n 番目の出現頻度の分岐先に分岐する確率					
ベンチマーク	1	2	3	4	5-8	9-
richards	84.3%	9.7%	4.2%	1.9%	0.0%	0.0%
eon	73.3%	18.4%	4.7%	3.6%	0.0%	0.0%
sjeng	38.4%	21.1%	9.7%	8.8%	17.4%	4.8%
gcc	51.0%	20.9%	12.9%	7.5%	6.1%	1.7%
perlbmk	48.8%	11.2%	9.0%	6.2%	11.7%	13.3%
割合	59.1%	16.3%	8.1%	5.6%	7.0%	3.9%
累積	59.1%	75.4%	83.5%	89.1%	96.1%	

ト) が追加されている．予測時には，命令アドレスの下位ビットで BTB と CB-BTP を引き，BTB より読み出した CB-BTP 移管ビットの値とタグの比較結果によって，BTB と CB-BTP のいずれかの出力を選択する．なお，ハイブリッド構成では，BTHT のエントリ数は BTB のエントリ数よりも少なくなるため，BTB 側で CB-BTP 移管ビットが 1 であっても，CB-BTP 上では対応するエントリが競合によって消えている場合もありうる．この場合，BTB 側の分岐先アドレスを予測値とする．

表の更新は，以下のような手順で行う．

（1） 分岐命令の命令アドレスの下位ビットで BTB と CB-BTP を引く．BTB 側でタグが一致しなければ，BTB に値を登録し，CB-BTP 移管ビットを 0 とする．

（2） (1) で BTB 側のタグが一致した場合，BTB から読み出した CB-BTP 移管ビットが 1 ならば，生成された分岐アドレスを用いて CB-BTP を更新する．この CB-BTP の更新は，3.1 節で示した手順で行われる．

（3） (2) で CB-BTP ビットが 0 の場合，生成された分岐先アドレスと BTB に保持されている分岐先アドレスを比較する．一致しない場合，CB-BTP 移管ビットを 1 とし，生成された分岐アドレスを用いて CB-BTP を更新する．

3.3 CB-BTP のハードウェア量とその削減の方向性

この節では，BTHT と PHT のハードウェア量について考察し，コスト削減の方向性について議論する．なお，以下のハードウェア量の議論において，命令アドレスの長さは 32bit とする．

BTHT については，必要なエントリ数については BTB と同程度であるが，1 エントリあたりの容量が大きいためにコストが高くなる．図 2 に示したように，BTHT の 1 エントリはタグ，複数の過去の分岐先アドレス，LRU 情報，履歴からなる．以下，各エントリのコストを示す．

- タグは BTB のタグと同様，$(32 - log_2$エントリ数$)$bit 必要となる．その量はエントリ数によって変化するが，おおむね，16bit から 24bit の幅に収まる．
- 過去の分岐先アドレスの保持については，4 個を保持すると仮定すると，128bit が必要となる．
- LRU 情報については，疑似 LRU を用いれば (分岐先アドレス保持数 − 1)bit が必要となる．
- 履歴については，(履歴長 × log_2分岐先アドレス保持数)bit となる．履歴は長い方が嬉しいが，履歴が PHT のインデクスとなる関係上，後述するように，履歴が 16bit 以上になると PHT のハードウェア量が非常に大きくなる．そのため，履歴は 16bit を上限と考える．

上記より，履歴やタグのビット数が多くなる構成でも，4 分岐先の保持の部分がハードウェア量の 75%以上を占めることになる．この部分のコスト削減については，文献 [9] に示されているような，分岐先アドレスの上位ビットの変化は少ないことを利用して，上位ビットを共有する方法が考えられる．

PHT については，1 エントリ当たりのハードウェア量は少ないが，履歴長に応じて指数的にエントリ数が増える点が問題となる．1 エントリのハードウェア量については，(飽和型カウンタのビット数 × 分岐先アドレスの保持数)bit となる．3.1 節で記したように，今回は 2bit 飽和型カウンタを用いたため，8 分岐先を保持しても 1 エントリは 16bit で済む．しかし，エントリ数が $2^{(\text{履歴長}\times log_2\text{分岐先アドレス保持数})}$ エントリとなるため，履歴が 16bit 以上となった場合，4 分岐先を保持した場合でも PHT のハードウェア量は 64KB 以上なってしまう．この履歴に対して指数的にエントリ数が増加する問題は分岐方向予測においても発生しており，その問題の緩和するために文献 [10] のようなパーセプトロンを用いる方法が提案されている．このような問題の緩和方法は，本手法にも適用できると考えられる．

4. 評　　価

この節では評価環境を提示し，その後，評価結果とその考察を示す．評価結果の項目では，まず in-order 実行による予備評価による，間接分岐によって，BTB による分岐先予測では不十分なベンチマークの抽出結果を示す．その後，それらのベンチマークにおける、CB-BTP の out-of-order 実行時における分岐先予測成功率と IPC 向上率を示す．

4.1 評価環境

提案機構を実装した，SimpleScalar Tool Set [11] の sim-bpred と sim-outorder を用いて提案機構の評価を行った．使用した命令セットは Alpha ISA である．ベンチマーク・プログラムには，SPECfp2006 と A C++ Benchmark Suite [12] の C++ ベンチマーク，および，SPECint2006 と SPECint2000 の C と C++ベンチマークの中から，Compac C Ver. 6.5 と Compaq C++ Ver. 7.1 を用いてコンパイルできたものを選択した．実行命令数は，4.1 節の in-order 実行による予測率の予備評価においては 10G 命令を実行し，4.2 節以降の out-of-order 実行のシミュレーションにおいては，最初の 1G 命令をスキップした後，1G 命令を測定に用いた。表 1 に，out-of-order 実行のシミュレーションにおいて仮定したプロセッサの構成を示す．

4.2 ベンチマークの予備評価

ベンチマーク・プログラムの中には，間接分岐の割合が少ないものや，動的には分岐先が 1 つに限定されてしまう間接分岐が多く含まれることもありうる．そのため，この節では，ベンチマーク・プログラム中の間接分岐の挙動の予備評価を行い，本評価の対象とするベンチマーク・プログラムの絞り込みを行う．なお，間接分岐の分岐先予測のうち，関数呼び出しからの復帰のための間接分岐 (以下，リターン分岐) については，リターン・アドレス・スタック (RAS: Return Address Stack) を利用することによって，容易かつ正確に予測可能である．よって，以下の間接分岐の分岐先予測の評価では，リターン分岐は除いてある．

表 2 に各プログラムにおける全分岐命令に対する全間接分岐の割合，全分岐命令に対するリターン分岐を除いた間接分岐の割合，リターン分岐除いた間接分岐の BTB による分岐先予測成功率を示す．ここでは，BTB は 4K エントリ/4-way を仮定している．表に示されるように，多くのベンチマークではリターン分岐以外の間接分岐がほどんど無かったり，リターン分岐以外の間接分岐でも BTB で十分に高い予測成功率を達成している．そのため，以後の評価では，表 2 中で太字で示されている，リターン分岐以外の間接分岐が多く，なおかつ，それらの間接分岐の BTB による予測成功率が低いベンチマークのみを対象とする．

選択したベンチマークの間接分岐における，静的な間接分岐の分岐先数を調べた所、多いベンチマークでも 265 個に過ぎないことが分かった。動的な挙動については、表 3 に実行時に n 番目の出現頻度の分岐先に分岐する確率を示す．表 3 より，選択したベンチマークにおいて，リターン分岐を除いた間接分岐では，最も出現頻度が高い分岐先のみを BTB に保持できていた場合でも，予測成功率は 59%に過ぎないことが分かる．また，理想的に複数の過去の分岐先を保持できれば，4 分岐先の保持で 89%の，8 分岐先の保持で 96%の予測成功率を達成できることも分かる。

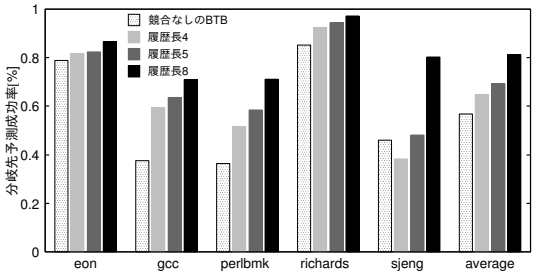

図 4　CB-BTP の予測成功率

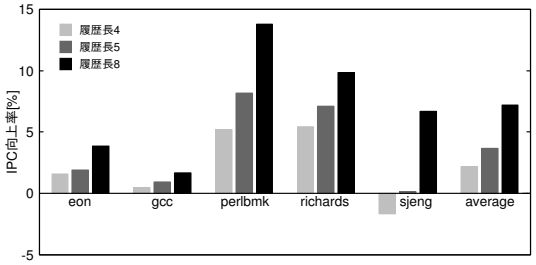

図 5　CB-BTP の IPC 向上率

4.3 CB-BTP のみ用いた場合の IPC の向上

この節では，CB-BTP の out-of-order 実行を行うプロセッサへの実装によって，IPC がどのよう変化するかを示す．比較対象としては，エントリ競合が起きない理想の BTB を比較対象とした．この節では，分岐先の予測は CB-BTP のみで行い，BTHT のエントリ数は 2K エントリとした．

BTHT の 1 エントリ当たりの分岐先アドレスの保持数は，1 履歴を記すのに必要なビット数の変化の境界にある，4 分岐先と 8 分岐先を候補とした．これら 2 つを同一のコストで予備評価を行ったところ，履歴は短くなっても，8 分岐先を保持した方が予測成功率は高くなるという結果を得た．よって，以後の評価で CB-BTP では 8 分岐先を保持するものを用いる．この場合，BTHT のコストは履歴長 4 で 73.7KB，履歴長 5 で 74.5KB となる．また，その時の履歴長は，PHT のサイズが現実的な大きさになる値として履歴長 4 と履歴長 5 を，将来的に履歴の表現を変更するなどとしてより長い履歴が使えることを前提とした値として履歴長 8 を採用する．PHT のコストは，履歴長 4 で 8KB，履歴長 5 で 64KB となる．

図 4 に out-of-order 実行における CB-BTP の分岐先予測成功率を示す．グラフの横軸はベンチマークとのその平均であり，グラフの縦軸は分岐先予測成功率である．ベンチマークごとの 4 本の棒グラフはそれぞれ，理想の BTB，履歴長 4,5,8 の時の予測成功率を示す．グラフより分かるように，sjeng の履歴長 4 を除いて理想の BTB よりも予測成功率は向上し，平均の予測成功率の向上は履歴長 4 で 8%ポイント，履歴長 5 で 13%ポイント，履歴長 8 で 25%ポイントとなった．

図 5 に理想の BTB の時の IPC に対する，CB-BTP の IPC

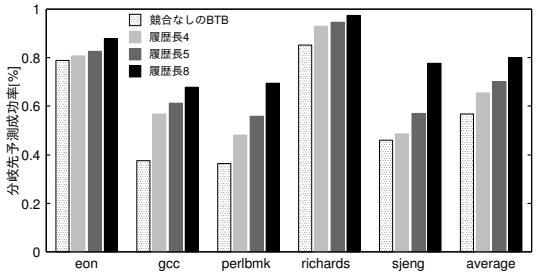

図 6　ハイブリッド予測器の予測精度

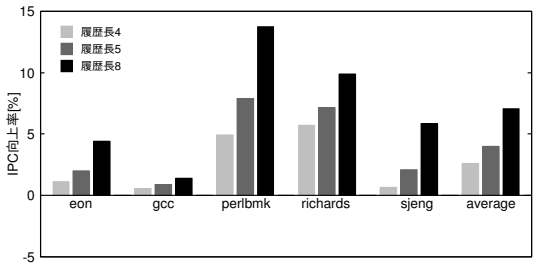

図 7　ハイブリッド予測器の IPC 向上率

向上率を示す．グラフの横軸はベンチマークとのその平均であり，グラフの縦軸は IPC 向上率である．ベンチマークごとの 3 本の棒グラフはそれぞれ，履歴長 4,5,8 の時の IPC 向上率を示す．図 4 で示されるように予測成功率が低下した sjeng の履歴長 4 は，IPC が低下してしまっている．それ以外の条件では IPC は向上し，特に，perlbmk では大きな向上が得られた．この perlbmk の大きな向上は，表 2 で示されるように perlbmk ではリターン分岐以外の間接分岐が多いため，間接分岐の分岐先予測成功率の向上の効果が大きく現れたからである．IPC 向上率の平均は，履歴長 4 で 2.2%，履歴長 5 で 3.7%，履歴長 8 で 7.2%となった．

4.4　ハイブリッド予測器による IPC の向上

この節では，ハイブリッド予測器の実装によって，プロセッサ性能がどのよう変化するかを示す．比較対象としては，エントリ競合が起きない理想の BTB を比較対象とした．予備評価により，CB-BTP のエントリ数を 256 以下にすると予測精度が低下するため，CB-BTP 側のエントリ数は 512 エントリとした．この傾向は，間接分岐数が最大のベンチマークで 265 個ということから予想される傾向と一致する．この場合，BTHT のコストは履歴長 4 で 18.6KB，履歴長 5 で 18.8KB となり，追加される BTB のコストは 13.3KB である．

図 6 にハイブリッド予測器によるの分岐先予測成功率を示す．グラフの横軸，縦軸，ベンチマークごとの 4 本の棒グラフは図 4 と同じである．全体的に，グラフは図 4 と同じ傾向を示すが，一部ベンチマークでは予測成功率が向上している．特に，sjeng の履歴長 4 と 5 では，大きな予測成功率の向上が見て取れる．また，逆に，CB-BTP のエントリ数が減少したことによってわずかながらも予測成功率が下がったものもある．

図 7 に理想の BTB の時の IPC に対する，ハイブリッド予測器の IPC 向上率を示す．グラフの横軸，縦軸，ベンチマークごとの 3 本の棒グラフは図 5 と同じである．図 5 では BTB よりも予測成功率が低下したことによって IPC が低下していた sjeng の履歴長 4 も性能向上に転じた．IPC の向上の平均は，履歴長 4 で 2.6%，履歴長 5 で 4.0%，履歴長 8 で 7.1%となり，短い履歴長では CB-BTP 単体の構成よりも大きな性能が得られた．

5. ま と め

レジスタ間接分岐命令の分岐先予測の問題に対し，本論文では，複数の分岐先予測を行う方法の 1 つとして，分岐命令のローカルの分岐先履歴をもとに分岐先予測を行う手法を提案した．提案する予測器による IPC の向上を評価した結果，履歴長 5 の場合は IPC は 4.0%向上するという結果を得た．コストを無視した履歴長 8 の場合は IPC は 7.1%向上するという結果が得られているため，提案手法は，履歴長の増加に対するコストの増大の問題を解決すれば，さらなる性能向上が得られると考えられる．このような履歴長の増加に対するコストの増大を緩和する手法として，分岐方向予測ではパーセプトロンを用いる方法が提案されており [10]，本手法の改良にも同様の手法を取れると考えられる．また，多数の分岐先を保持するために増加するコストの削減については，分岐先アドレスの上位ビットを共有する方法 [9] を使うことができると考えられる．

将来の課題としては，前述のハードウェア・コスト削減の他に、他の間接分岐の分岐先予測手法との比較やそれらとのハイブリッド構成の評価、リターン分岐以外の間接分岐の多いベンチマークでの評価が挙げられる。

謝辞　本研究の一部は日本学術振興会科学研究費補助金基盤研究 S(課題番号 16100001) による．

文　献

[1] Chang, P.-Y., et al.: Target Prediction for Indirect Jumps, *ISCA-24*, pp. 274–283 (1997).
[2] Driesen, K. et al.: The Cascaded Predictor: Economical and Adaptive Branch Target Prediction, *MICRO-31*, pp. 249–258 (1998).
[3] Kim, H., Joao, et al.: VPC Prediction: Reducing the Cost of Indirect Branches via Hardware-Based Dynamic Devirtualization, *ISCA-34*, pp. 424–435 (2007).
[4] 豊島隆志ら: レジスタ間接分岐ターゲット・フォワーディング, SACSIS2006, pp. 325–332 (2006).
[5] Lipasti, M. H. et al.: Exceeding the Dataflow Limit via Value Prediction, *MICRO-29*, pp. 226–237 (1996).
[6] Chen, T.-F. et al.: Effective Hardware-Based Data Prefetching for High-Performance Processors, *IEEE Trans. on Computers*, Vol. 44, No. 5, pp. 609–623 (1995).
[7] Sazeides, Y. et al.: The Predictability of Data Values, *ISCA-24*, pp. 248–258 (1997).
[8] 片山清和ら: 値予測を利用した分岐予測機構,, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol. 42, No. SIG 12(HPS 4), pp. 22–36 (2001).
[9] 小林良太郎ら: 2 レベル表方式による分岐先バッファ, 情報処理学会論文誌, Vol. 41, No. 5, pp. 1351–1359 (2000).
[10] Jimenez, D. A. et al.: Dynamic Branch Prediction with Perceptrons, *HPCA-7*, pp. 197–206 (2001).
[11] Burger, D. et al.: The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-97-1342, University of Wisconsin-Madison Computer Sciences Dept. (1997).
[12] OOCSB: Object-Oriented Compilers at UCSB –A C++ Benchmark Suit, http://www.cs.ucsb.edu/ urs/oocsb/.