Vol. 43 No. 9 情報処理学会論文誌 Sep. 2002

パス情報を用いた分岐フィルタ機構

三輪 忍[†1] 福山智久[†1] 嶋田創[†1]
五島正裕[†2] 中島康彦[†3]
森眞一郎[†4] 富田眞治[†1]

PHT (Pattern History Table) における破壊的競合を抑制する方法にフィルタ機構がある．フィルタ機構では，強 偏向の分岐命令は PHT を使用しないようにすることで，PHT の予測ミス率を低下させる．一方，パス情報が予測の手がかりになることが最近になって分かってきた．そこで本稿では，フィルタ機構においてパス情報を利用する手法を提案する． 強偏向のパスは PHT を使用しないようにすることで，予測ミス率の低下を狙う． 本手法を Global Perceptron Predictor に適用した場合，平均 0.14 % ミス率が低下した．特に，go においては 0.7 % のミス率低下が見られた．また，本手法を Path Based Predictor, Path Trace Predictor に適用した場合でも，go において 0.6~0.7% のミス率低下が見られた．

Branch Filtering Mechanism with Path Trace

Shinobu Miwa,[†1] Tomohisa Fukuyama,[†1] Hajime Shimada,[†1]
Masahiro Goshima,[†2] Yasuhiko Nakashima,[†3] Shin-ichiro Mori[†4]
and Shinji Tomita[†1]

Branch filter mechanism is a method which reduces destructive aliasing on PHT (Pattern History Table). This improves the misprediction rate not to use PHT for the branches with strong tendencies. Otherwise, it proves recently that path traces are hint for branch predictions. So, we propose branch filter mechanism with path traces. It is supposed that this mechanism improves the misprediction rate not to use PHT for the path trace with strong tendencies. When this mechanism is impremented on Global Perceptron Predictor, the average misprediction rate is reduced 0.14%. Especially, the misprediction rate of *go* is reduced 0.7%. And, when proposal mechanism is impremented on Path Based Predictor, on Piecewise Linear Predictor, and on Path Trace Predictor, the misprediction rate of *go* is reduced 0.6~0.7%.

1. はじめに

近年では，マイクロプロセッサの高クロック化に伴い，パイプラインは深化傾向にある[9]．その一方，パイプラインの深化は，分岐予測ミス・ペナルティの増大をもたらす．分岐予測ミス時のペナルティは，命令パイプラインの命令フェッチ・ステージから実行ステージまでのサイクル数で与えられる．そのため，**分岐予測**のヒット率の改善は重要な課題となっている．

これまでに数多くの分岐予測器が提案されてきたが，その大半は飽和型カウンタを用いた方式であった．通常，2 bit (以下 b とする) のカウンタをエントリとする，**PHT** (Pattern History Table) と呼ばれるテーブルが用いられる．最も単純な方式では，PHT のインデクスに，分岐命令のアドレスの下位の一部を用いる．

最も代表的な **g-share** 分岐予測器[6] では，PHT のエントリを指し示すインデクスの一部に**グローバル分岐履歴**(global branch history) を加えることで，ヒット率の向上を狙っている．

しかし，インデクスの一部にグローバル分岐履歴を用いる方式では，同一の分岐に対しても，出現するグローバル分岐履歴のパタンの数だけエントリを更新することになる．その結果，競合がなければ予測ヒットするが実際にはミスする競合，すなわち，**破壊的競合** (destractive aliasing)[1] の発生確率が上昇してしまう．

破壊的競合による予測精度の低下を防ぐ方法に，フィルタ機構[2] がある．フィルタ機構では，Taken，Untaken のどちらかに強く偏向している分岐命令を予測器による予測の対象から外すことで，予測ヒット率を向上さ

†1 京都大学
Kyoto University
†2 東京大学
University of Tokyo
†3 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
†4 福井大学
Fukui University

せている．

パーセプトロンを用いた分岐予測器

予測ヒット率を向上させるには，一般に，グローバル分岐履歴は長い方がよい．ところが，g-share では，グローバル分岐履歴長の指数に比例するメモリ量が必要であった．分岐予測器に割けるメモリ量は数 KByte (以下 KB とする) からせいぜい数百 KB 程度で，その範囲内では十数個分の履歴しか扱えない．

Jiménez らはそれに対して，パーセプトロン[8] を用いた分岐予測器を提案している[3]．パーセプトロンを用いた分岐予測器では，必要なメモリ量は，グローバル分岐履歴長に比例する．結果，同メモリ量で比較的長い履歴を用いることができ，既存の方式よりも高い予測ヒット率を示している．

ただし，パーセプトロンを用いた分岐予測器は多くの計算量を必要とする．予測に際しては，整数の加減算が，履歴長に比例した回数行われる．そのため，履歴長は，フェッチしてから実行するまでのパイプライン・ステージ段数に支配される．

そこで，予測処理をパイプライン化する方法[4] が提案された．パイプライン化された分岐予測器では，分岐命令がフェッチされる度に，その時の分岐命令のアドレスによって重みを求め，それを累積する．予測する命令がフェッチされると，累積値を使って予測する．

パス情報と分岐予測

予測処理のパイプライン化は大きなパラダイムシフトとなった．それまでは，予測の手がかりとなる情報と言えば，(ローカル/グローバル) 履歴と予測する分岐命令のアドレスであった．それがパイプライン化によって，分岐命令のアドレスの履歴，すなわち，**パス情報**も強力な手がかりとなることが分かってきた．

現在は，パス情報を利用することで予測ヒット率を向上させた分岐予測器が，数多く提案されている[5][10]．

本稿では，フィルタ機構においてパス情報を利用する方法を提案する．上述のように，フィルタ機構では，強偏向の分岐命令は予測器を使用しないようにすることで，予測ヒット率を向上させる．それと同様，強偏向のパスは予測器を使用しないようにすれば，より高い予測ヒット率を実現できるだろう．

以下まず次章では，フィルタ機構，および，パス情報を利用した分岐予測器について詳しく述べる．続く 3 章では，予備評価として強偏向のパスがプログラム中にどの程度存在するかを示し，それらをフィルタリングする機構を提案する．評価は 4 章で行う．

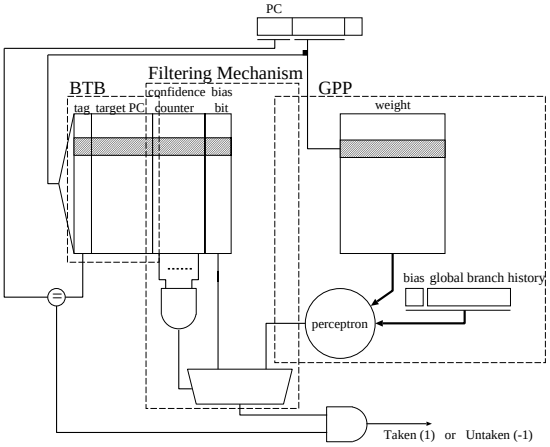

図 1 Filiter 機構
Fig. 1 Branch filter mechanism.

2. 関 連 研 究

本章では，破壊的競合を緩和する方法としてよく知られた，フィルタ機構についてまず述べる．次いで，近年主流になりつつある，パス情報を利用する分岐予測器について述べる．

2.1 フィルタ機構

フィルタ機構は，予測器を使用する対象から強偏向の分岐命令を外すことで，予測ヒット率を向上させる方法である．プログラム中，always Taken, always Untaken の分岐命令は全分岐命令の約 50% を占めていることが知られている[2]．このような分岐命令は予測器を使用しないようにすれば，PHT における破壊的競合を減らせる．

フィルタ機構の構成を図 1 に示す．図では，予測器には，後述する **GPP** (Global Perceptron Predictor) を用いている．

フィルタ機構は，**BTB** (Branch Target Buffer) の各エントリに 2 つのフィールドを追加することで実現される．1 つは**確信度カウンタ**と呼ばれる 2~3b の飽和型カウンタで，これにより分岐結果の偏りを検出する．その初期値は 0 とする．もう 1 つは**バイアス・ビット**と呼ばれる 1b のフィールドで，これは偏向している方向を (Taken を 1，Untaken を −1 で) 表す．

フェッチした分岐命令が BTB にヒットした時，フィルタ機構を用いた予測は，次のようにして行う．まず，ヒットしたエントリの確信度カウンタを参照する．結果，確信度カウンタが：

全ビット 1 だった場合 バイアス・ビットの値を予測結果とする．分岐予測器は使用しない．

それ以外の場合 分岐予測器 (図では GPP) を使用し，

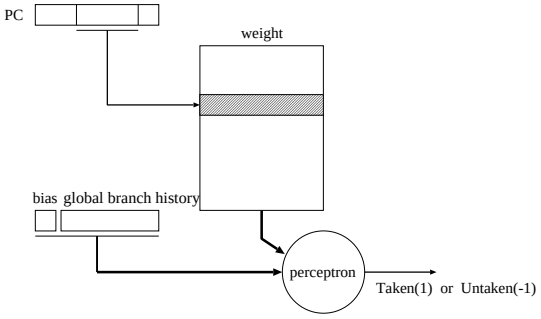

図 2 GPP (Global Perceptron Predictor)
Fig. 2 GPP (Global Perceptron Predictor)

その結果を予測結果とする

なお，BTB にミスした時は，分岐予測器を使用せずに Untaken を返す[7].

フィルタ機構の更新は以下のようにして行う：

- ヒットしたエントリのバイアス・ビットと分岐結果が一致した場合，確信度カウンタをインクリメントする
- 一致しなかった場合，バイアス・ビットを反転し，確信度カウンタを 0 にする

なお，分岐予測器の更新はそれを使用した時のみ行う．

例えば，3b の確信度カウンタをもつフィルタ機構で，always Taken の命令を予測するとしよう．この命令が最初にフェッチされた時，カウンタは 0 であるため，予測器が使用される．そして，バイアス・ビットに 1 がセットされ，カウンタは 1 になる．2 回目も，カウンタはまだ最大値になっていないため，予測器が使用されカウンタが 1 増える．そして，8 回目になると，カウンタは最大値になっているため，バイアス・ビットが使用される．以降，この命令の予測には，全てバイアス・ビットが使用されることとなる．

このようにして，フィルタ機構は，強偏向の分岐命令が予測器を使用するのを防ぐ．

2.2 パス情報を利用する分岐予測器

冒頭で述べたように，近年では，パス情報を利用した分岐予測器が主流となりつつある．本節では，そうした予測器である，PBP (Path-Based Predictor), および, PTBP (Path Trace Branch Predictor) について述べる．

これらの予測器は，GPP (Global Perceptron Predictor) をベースに発展してきた．そこで，まずは GPP について述べることにしよう．GPP 自体は，パス情報を利用していない点に注意されたい．

2.2.1 GPP (Global Perceptron Predictor)

図 2 に，GPP の構成を示す．予測においてはパーセプトロンが中心的な役割を果たす．パーセプトロンに

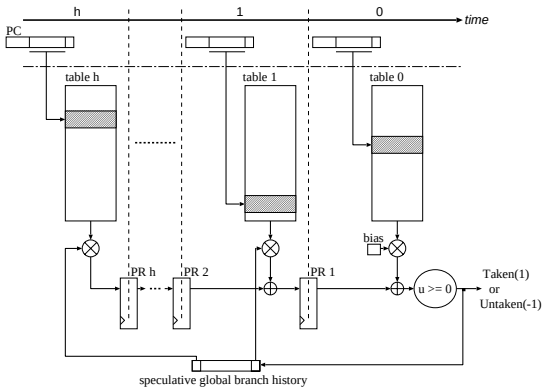

図 3 PBP (Path Based Predictor)
Fig. 3 PBP (Path Based Predictor)

グローバル分岐履歴を入力して分岐予測を行う．なお，入力として，グローバル分岐履歴だけでなくローカル分岐履歴も用いた，Global /Local Perceptron Predictor も提案されている．

重みセット (weight set) は複数用いる．それらは，PC をハッシングしたものをインデクスとするテーブルに格納されている．予測に際しては，分岐命令のアドレスを用いてテーブルにアクセスする．ただし，g-share などとは異なり，インデクスの生成にはグローバル分岐履歴を用いない．分岐命令に対して 1 つの重みセットが選ばれることになる．この重みセットをパーセプトロンの重みとし，グローバル分岐履歴を入力として，予測を行うのである．

入　　力

入力は 1(Taken)/-1(Untaken) で与えられる．

重　　み

8 ビット符号付き整数型で表される．入力も 1/-1 であるために，積和の計算が加減算に帰着できる．

出力と予測

積和が 0 以上の時 1 (Taken)，そうでなければ -1 (Untaken) を出力する．

上述のように，パーセプトロンの入力の長さは履歴長☆ によって与えられる．そして，その分の加減算が，予測に際して行われる．したがって，履歴が長いと，予測結果が実行ステージに間に合わなくなってしまう．

そこで，予測処理をパイプライン化した PBP が提案された．それでは，次は PBP について述べることにしよう．

2.2.2 PBP (Path-Based Predictor)

前項で述べたように，GPP のボトルネックは，履歴

☆ 正確にはバイアスも含め (履歴長+1)

長分の整数の加減算であった．そこで，予測処理をパイプライン化し，1 ステージに行うのは加減算 1 回とすることで，長い履歴を用いた予測を可能にする．

図 3 に PBP の構成を示す．PBP は，GPP のテーブルを履歴方向に分割した構成となっている．すなわち，PBP のテーブルのエントリは，重みセットではなく重みである．各テーブルを履歴順にテーブル $0, 1, \cdots, h$ とする．テーブル間には，パイプライン・レジスタ PR i ($i = 1, \cdots, h$) が設けられている．

予測は投機的に行う．例えば，履歴長が h の場合，ある分岐命令がフェッチされると，その h 個先の分岐命令の予測を開始する．したがって，ある分岐命令がフェッチされた時点では，パイプライン・レジスタによって区切られた h 個の区間において，その分岐命令から h 個先の分岐命令までの予測が行われている．

分岐命令がフェッチされると，PC をハッシングしたものをインデクスとして，以下の処理を行う：

- **バイアス**[8] を読み出す．それを PR 1 の累積値に加え，予測結果とする．
- テーブル i の重みを読み出す．それと PR $i+1$ の累積値とを加算，または，減算し，結果を PR i へ格納する．加算するか減算するかは，その時の予測結果☆ によって決定する．

このように，PBP では，履歴長によらず，PR 1 の値とバイアスとの加算 1 回で予測が行える．

また，この方法の重要な点はパス情報，すなわち，分岐命令のアドレスの履歴が利用されている点である．図に示すように，時刻 0 の分岐命令の予測に使用されるのは，その時の PC だけではない．その命令の積和を求める過程で，時刻 $1 \sim h$ の PC も使用される．

性　　能

同メモリ量 (8KB) の Global /Local Perceptron Predictor に比べ，SPEC CINT 2000 ベンチマークにおいて予測ヒット率が平均 0.4%向上するという結果が出ている．

このように，パイプライン化されたにも関わらず，PBP の予測ヒット率は向上する．このことは，パス情報が有効な役割を果たしたと言えよう．

2.2.3 PTBP (Path Trace Branch Predictor)

最後に，パス情報をより効果的に利用した，PTBP について述べる．

前項で述べた PBP では，アドレスを独立したものとして扱っていた．個々のアドレスは，それぞれ，異なるテーブルのインデクシングに利用される．そうして独立に求めた重みを加算し，パス情報を 1 つの値へとマッピングする．

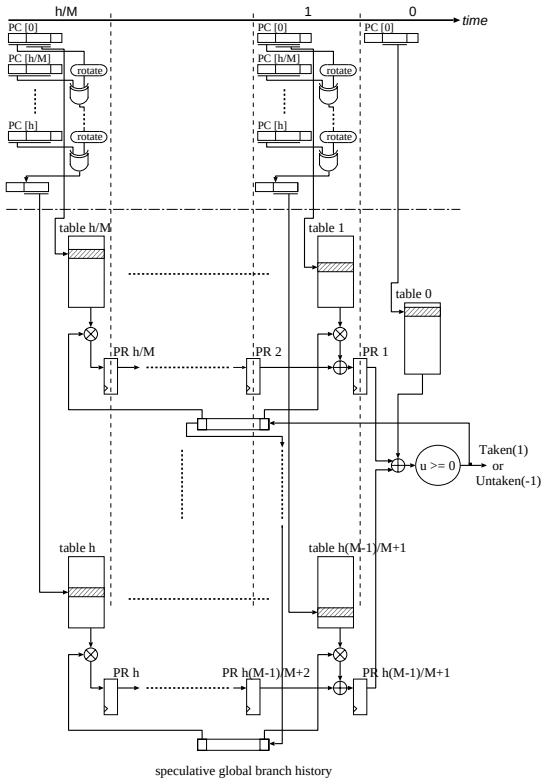

図 4 PTBP (Path Trace Branch Predictor)
Fig. 4 PTBP (Path Trace Branch Predictor)

それに対し，PTBP は，アドレス間の関連性をより直接的に利用する．具体的には，単一アドレスでテーブルをインデクシングするのではなく，複数アドレスの排他的論理和によってインデクシングする．

アドレス間の関連性はなるべく多く利用した方がよい．そこで，以下で詳しく述べるように，PTBP では予測の開始を遅らせている．

図 4 に PTBP の構成を示す．PTBP は，PBP を時間方向に M 等分し，分割したパイプラインを空間方向に並列に並べた構造をしている．ただし，単に並列に並べたものとは，以下のようにインデクシング方法が異なる：

テーブル h/M まで ($1 \sim h/M$) PBP と同じく，その時の PC をハッシングしたものをインデクスとする

テーブル h/M 以降 ($h/M+1 \sim h$) その時の PC からその履歴の PC までの排他的論理和をハッシングしたものをインデクスとする．ただし，排他的論

☆ 予測時にはまだ分岐結果が確定していないため，分岐結果ではなく予測結果を用いる．予測結果と分岐結果が異なった場合，当然リカバリが行われるが，その処理は本稿では述べない．

理和は，20b のローテーション・シフトを繰り返しながら計算する．これは，パス情報に同一 PC が含まれる時，その排他的論理和が 0 となるのを防ぐためである．簡単のため，排他的論理和をとる PC は h/M 個おきとする．

例えば，テーブル h のインデクシングには，その時の PC，h/M 個前の PC，$2h/M$ 個前の PC，$\cdots$，h 個前の PC が利用される．このように，PTBP は，アドレス間の関連性を利用する．

性　　能

後述するように，同メモリ量 (128KB) の PBP に比べて，SPEC CINT 95 ベンチマークにおいて予測ヒット率が平均 0.63% 向上するという結果が出ている．

このように，予測ヒット率を改善する上でパス情報は効果的である．

3. パス情報を用いた分岐フィルタ機構

2.1 で述べたように，フィルタ機構は，強偏向の分岐命令が予測器を使用するのを防ぐことで，予測ヒット率を向上させる手法であった．一方，2.2.2, 2.2.3 で述べたように，パス情報が予測精度向上のための強力な手がかりとなることが最近になってわかってきた．

そこで，本稿では，フィルタ機構においてパス情報を利用する手法を提案する．ある分岐命令をある順序で通過した場合，その終端の分岐命令が always Taken, always Untaken になることがある．そうした，言わば，強偏向のパスが予測器を使用するのを防げば，破壊的競合が減り，予測ヒット率は向上するだろう．

以下まず次節で，予備評価として SPEC CINT 95 の 8 本のベンチマークにおいて，強偏向のパスがどの程度存在するかを示す．そして，それらをフィルタリングする機構を次々節で提案する．

3.1 予 備 評 価

前述のように，プログラム中の分岐命令の約半分は強偏向の分岐命令である．すなわち，always Taken，または，always Untaken な分岐命令が半分を占める．そのため，フィルタ機構は高い効果を発揮する．

これと同様のことが，パスについても言える．パスの多くが強偏向であれば，パス情報を利用したフィルタリングは，高い効果が期待できる．

SPEC CINT 95 における強偏向のパスの割合を**図 5** に示す．グラフの横軸は分岐命令のアドレスの履歴数，すなわち，**パス長**を，縦軸は強偏向のパスの割合を表す．ここで，強偏向のパスの割合とは，always Taken, または，always Untaken なパスの出現回数を，全パスの出現回数で割ったものである．なお，図は BTB 512-set, 4way 時の結果で，BTB にミスするパスは分子/分母共に含めていない．測定には，SimpleScalar ツールセット (ver. 3.0) の sim-bpred シミュレータを用いた．

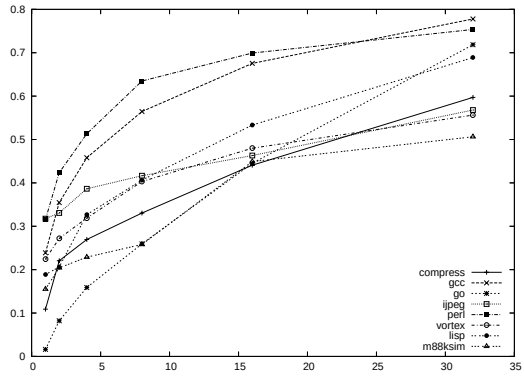

図 5 各ベンチマークにおける強偏向のパスの出現率
Fig. 5 Rate of the paths which is always taken or which is always untaken

グラフより，パス長 1，すなわち，always Taken な分岐命令の割合は，最も少ない go で 1.6%，最も多い ijpeg で 31.7% である．どのベンチマークにおいても，パス長が長くなる程，強偏向なパスの割合は増加する．パス長が 32 の時，強偏向なパスの割合は，最も少ない m88ksim で 50.6% 最も多い gcc で 77.8 % になる．

また，グラフからわかるように，go を除き，強偏向なパスの割合はパス長に対して対数的に増加する．そして，パス長 4~ 16 程度でほぼ上限 (4 ~ 7 割) に達する．言い換えると，パス長が 4~ 16 あれば，4 ~ 7 割のパスはフィルタリングできる．

3.2 パス情報を用いたフィルタ機構

前節で，強偏向なパスはかなりの割合で存在することがわかった．そこで本節では，そのようなパスをフィルタリングする機構を提案する．

パス情報を用いたフィルタ機構の構成を**図 6** に示す．パス情報を用いたフィルタ機構は，通常のフィルタ機構 (**図 1**) と同様，確信度カウンタとバイアス・ビットをエントリとするテーブルによって構成する．ただし，通常のフィルタ機構とは以下の 2 点が異なる：

(1) 分岐命令のアドレスではなく，アドレスの履歴によってインデクシングする．アドレスの履歴は，パス長 (n) 分のアドレスの排他的論理和によって表現する．PTBP と同様の理由により，20b の ローテーション・シフトを繰り返しながら排他的論理和を計算する．一方，PTBP とは異なり，排他的論理和は n 個分全てのアドレスのそれとする．

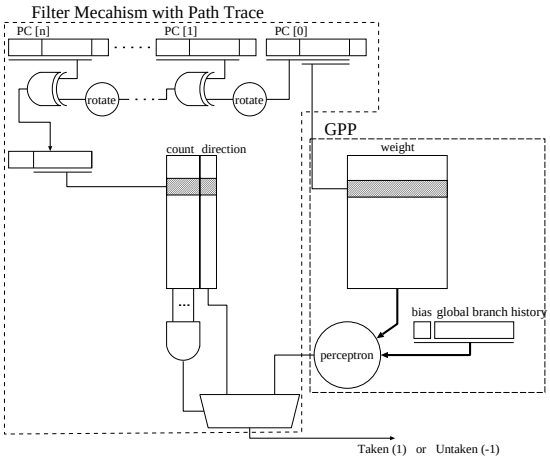

図 6 パス情報を用いたフィルタ機構
Fig. 6 Filtering mechanism with path trace.

(2) タグは存在しない．すなわち，エントリには必ずヒットする．

ヒットした後の動作は，通常のフィルタ機構と同様とする．すなわち，ヒットしたエントリの確信度カウンタを参照し：

全ビット 1 だった場合 バイアス・ビットの値を予測結果とする．分岐予測器は使用しない．

それ以外の場合 分岐予測器を使用し，その結果を予測結果とする

また，更新も通常のフィルタ機構と同様とする．すなわち：

- バイアス・ビットと分岐結果が一致した場合，確信度カウンタをインクリメントする
- 一致しなかった場合，バイアス・ビットを反転し，確信度カウンタを 0 にする

例えば，パス長 4，3b カウンタのパス情報を用いたフィルタ機構で，always Taken な $A_3 \rightarrow A_2 \rightarrow A_1 \rightarrow A_0$ (A_i ($i = 0, \cdots, 3$) はアドレス) というパスをフィルタリングするとしよう．A_0 がフェッチされた時，まずは，過去 4 つ分のアドレスの排他的論理和 $(A_3 \oplus A_2 \oplus A_1 \oplus A_0)$ を求める．そして，それをハッシングしたものをインデクスとしてテーブルを参照する．最初は確信度カウンタが 0 なので，予測器を使用することになる．そして，バイアス・ビットに 1 をセットしてカウンタをインクリメントする．次にこのパスが現れた時も同様に，排他的論理和によってカウンタを参照する．やはり，カウンタは最大値に達していないため，2 回目も予測器を使用し，カウンタをインクリメントする．やがて，このパスが 8 回目に現れると，カウンタは最大値に達しているため，今度はバイアス・ビットにより予測する．以降，このパスが現れても，予測器は使用されなくなる．

このように，提案手法は，強偏向のパスをフィルタリングする．

プログラム	入力セット
129.compress95	10000 q 2131
126.gcc	amptjp.i
099.go	9 9
132.ijpeg	vigo.ppm
134.perl	primes.in
147.vortex	persons 250
130.li	train.lsp
124.m88ksim	ctl.raw

表 2 SPEC CINT95 ベンチマーク・プログラム
Table 2 SPEC CINT 95 benchmark programs.

4. 評　　価

提案手法を SimpleScalar ツールセット (ver. 3.0) の sim-bpred シミュレータに対して実装し，評価を行った．以下ではその結果を述べる．

4.1 評 価 環 境

以下の 9 通りのモデルを実装し，予測ヒット率を測定した：

(1) パス情報を利用しない予測器の代表例として GPP．また，パス情報を利用した予測器の代表例として PBP, PTBP の 2 モデル．(wo F)
(これら 3 つのモデルの予測ヒット率は，2.2 で述べたように，GPP< PBP< PTBP の関係にある．なお，PTBP のパイプラインは 4 本とする．)

(2) (1) のフィルタ機構付 (w F)

(3) (1) のパス情報を利用したフィルタ機構付 (w PTF)

表 1 に，測定に使用した，各メモリ量 (Hardware Budget) の範囲内で最適化された履歴 長とテーブル・サイズを示す．なお，学習の閾値[3)–5)] は，履歴長を h として，全モデル $2.1 * (h + 1)$ とした．

測定には，**表 2** に示す SPEC CINT95 の 8 本のプログラムを使用した．

BTB は 512set, 4way とした．また，BTB にミスした時は予測器を使用せず，Untaken を返すようにした[7)]．

フィルタ機構，および，パス情報を用いたフィルタ機構の確信度カウンタは，それぞれ，3b とした．☆

☆ 普通，フィルタ機構は BTB を拡張して実装するため，そのエントリ数は BTB に等しくなる．しかし，BTB とは独立な機構であるパス情報を用いたフィルタ機構と同一エントリのもとで比較するため，本研究ではフィルタ機構も独立に実装した．なお，本来は BTB を拡張して実装されるものであるから，独立に実装し

Hardware	wo Filter		w Filter (4K ent.)		w PTBF (4K ent.)		w PTBF (2K ent.)		w PTBF (1K ent.)	
Budget	History	Size	History	Size	History	Size	History	Size	History	Size
4KB	28	128	16	128	16	128	24	128	28	128
8KB	28	256	24	256	24	256	28	256	28	256
16KB	28	512	28	512	28	512	28	512	28	512
32KB	28	1024	28	1024	28	1024	28	1024	28	1024
64KB	60	1024	60	1024	60	1024	60	1024	60	1024
128KB	60	2048	60	2048	60	2048	60	2048	60	2048

表 1 最適化された履歴長とテーブル・サイズ
Table 1 Tuned history length and table size for each predictor.

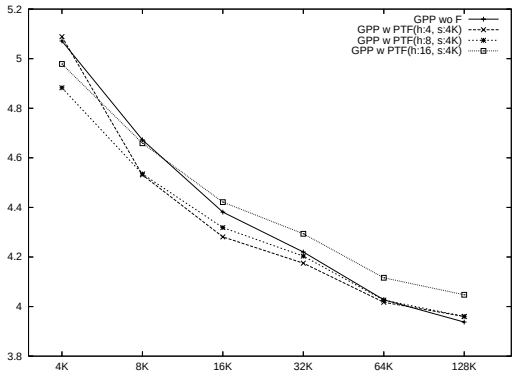

図 7 パス長に対する予測ミス率
Fig. 7 Misprediction rate for the length of the path

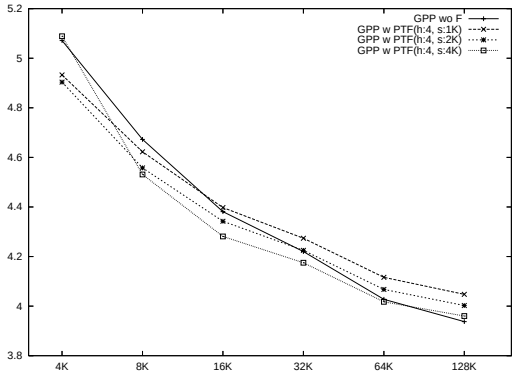

図 8 パス情報を用いたフィルタ機構の容量に対する予測ミス率
Fig. 8 Misprediction rate for the size of filter mechanism with path trace

4.2 結　　果

パス長に対する性能

図 7 に，パス情報を用いたフィルタ機構のパス長を 4, 8, 16 と変化させた時の，8 本のベンチマークの平均予測ミス率を示す．横軸はメモリ量，縦軸は平均予測ミス率を表す．フィルタ付のモデルのメモリ量には，フィルタ自体のそれも含まれている点に注意されたい．フィルタは 4K エントリとし，分岐予測器には GPP を使用した．

グラフより，メモリ量 8~32 KB の範囲ではパス長 4 のモデルが最も性能が良い．特に 8KB 構成時には，通常の GPP の平均予測ミス率が 4.67 % だったのに対し，パス情報を用いたフィルタ機構付 GPP では 4.53%と 0.14 % 改善した．

パス長を長くしても性能が向上しないのは，インデクシングに用いるパス情報が，以下のように情報圧縮されているためと考えられる：

(1) パス長分のアドレスを連結するのではなく，排他的論理和をとっている
(2) 排他的論理和は 20b のローテーション・シフトを繰り返しながら計算するが，実際にインデクシングに利用するのは，その内の数ビット (4K エントリのテーブルでは 12b) である

容量に対する性能

図 8 に，パス情報を用いたフィルタ機構のテーブルのエントリ数を 1K, 2K, 4K と変化させた時の，8 本のベンチマークの平均予測ミス率を示す．グラフの見方は図 7 と同様である．パス長は 4 とし，分岐予測器には GPP を使用した．

グラフより，4K エントリのパス情報を用いたフィルタ機構付 GPP が最も性能が良い．「パス長に対する性能」の項でも述べたように，8KB 構成時には，通常の GPP に比べ 0.14 % 改善する．

フィルタ機構との比較

9 個のモデルの平均予測ミス率を**図 10** に示す．グラフの見方は図 7, 8 と同様である．パス長は 4 とし，フィルタ，パス情報を用いたフィルタ共にエントリ数は 4K とした．

グラフからわかるように，どの分岐予測器においても フィルタ機構付よりパス情報を用いたフィルタ機構付の方が性能が良い．特に，4 KB 構成時においては，フィルタ機構付 GPP の予測ミス率が 5.28% だったのに対し，パス情報を用いたフィルタ機構付 GPP は 5.09 % と 0.19 % 改善した．

たとは言え，次節の結果ではタグの容量は考慮していない．

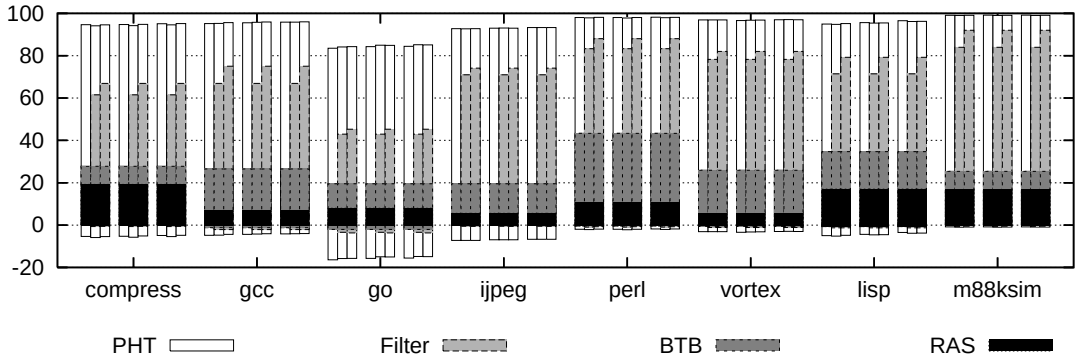

図 9 予測ヒット/ミス率の内訳 (8KB 構成時)
Fig. 9 Breakdown of hit /miss rate with a 8 KB hardware budget.

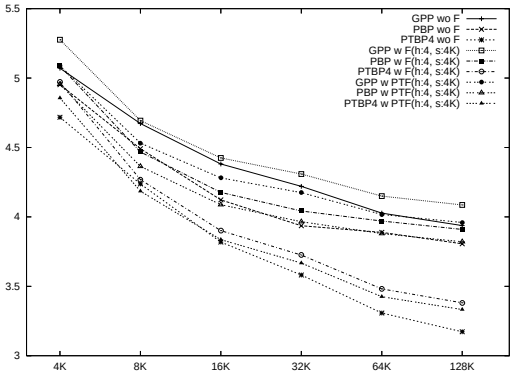

図 10 各モデルの予測ミス率
Fig. 10 Misprediction rate of each models.

図 9 に，8KB 構成時における，ベンチマーク毎の予測ヒット率の内訳を示す．1 つのベンチマークにつき，3 つ組の棒グラフが 3 セット並んでいる．3 つ組は，左から順に，フィルタ無し，フィルタ機構付，パス情報を用いたフィルタ機構付を表す．また，3 つのセットは，左から順に，GPP，PBP，PTBP である．縦軸の正方向はヒット率を，負方向はミス率を表す．ヒット/ミスは，それぞれ，以下の 4 つに分けられている：

PHT 分岐予測器使用時のヒット/ミス

Filter フィルタ使用時のヒット/ミス

BTB BTB ミスにより Untaken と予測した時のヒット/ミス

RAS return address stack 使用時のヒット/ミス

グラフより，どのベンチマークにおいても，パス情報を用いたフィルタ機構はフィルタ機構よりもフィルタリング率/精度共に良い．特に gcc においては，RAS, BTB, Filter によってヒットした分岐の割合の合算値が，フィルタ機構付では 66.9% だったのに対し， パス情報を用いたフィルタ機構付では 75.0% と 8.1% 増加した．一方，ミスした場合のそれは，どちらも 2.10% と変化していない．

分岐予測器に対する性能

図 10 において，パス情報を用いたフィルタ機構「無し」のモデルと「有り」のモデルを比較するとわかるように，パス情報を利用しない予測器である GPP では，パス情報を用いたフィルタ機構の効果が見られる．特に，8KB ~ 32KB の区間では,「有り」のモデルの平均予測ミス率が「無し」のそれを下回っている．

また，ベンチマーク毎の性能 (図 9) を見ると，compress を除き,「有り」の予測ヒット率が「無し」のそれを上回っている．特に，go においては,「有り」が 84.3%,「無し」が 83.6% と 0.7% 改善している．

一方，パス情報を用いたフィルタ機構をパス情報を利用する予測器に適用した場合は，その効果は薄い．PBP は 8~16 KB 構成時，PTBP は 8 KB 構成時においてのみ,「有り」のモデルの平均予測ミス率が「無し」のモデルのそれを下回る．

ベンチマーク毎の性能 (図 9) を見ると，GPP と同様，go における性能向上が強く影響していることがわかる.「有り」の予測ヒット率は，PBP，PTBP がそれぞれ 84.9%，85.1%,「無し」が 84.3%，84.4% と 0.6~0.7% 改善している．

破壊的競合の割合

図 11 に，8KB 構成時の破壊的競合の割合を示す．3 つ組，3 セットの棒グラフは， 図 9 と同様，9 つのモデルに対応する．すなわち，3 つ組は，左から順に，フィルタ無し，フィルタ機構付，パス情報を用いたフィルタ機構付きを，3 セットは，左から順に，GPP, PBP, PTBP を表す．

ここで，破壊的競合の割合は，分岐命令のそれでは

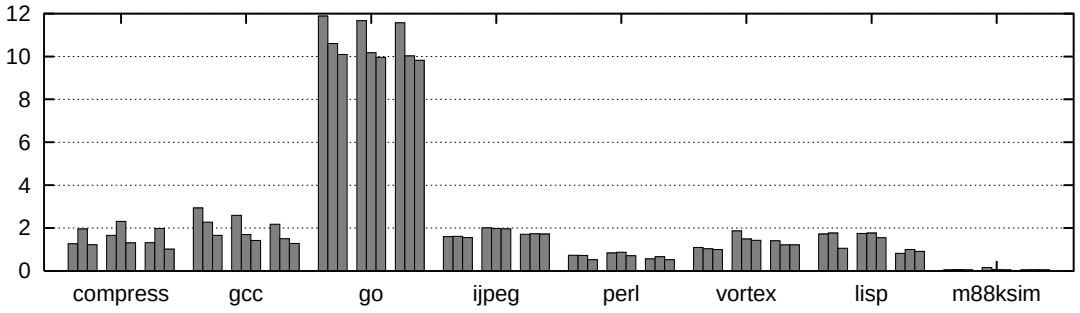

図 11 破壊的競合の割合 (8KB 構成時)
Fig. 11 Destructive aliasing rate with a 8 KB hardware budget.

なく，パスのそれを表している点に注意されたい．

例えば，$A_3 \rightarrow A_2 \rightarrow A_1 \rightarrow A_0$ というパスと $A_3 \rightarrow A_2 \rightarrow A_1' \rightarrow A_0$ というパスが PHT の同一エントリを使用したとしよう．その場合，分岐命令のアドレスは同じ (A_0) であるから，分岐命令は競合していない．したがって，2 つのパスに別々のエントリが割り当てられた場合と予測結果が異なり，かつ，予測ミスしたとしても，破壊的競合とは呼ばない．

一方，上述の例では，分岐命令のアドレスの履歴は異なる (A_1 と A_1') から，パスは競合している．そのため，別々のエントリが割り当てられた場合と予測結果が異なり，ミスした場合は，破壊的競合と数える．

図 11 より，破壊的競合の割合は，ほとんどのベンチマーク，予測器において，パス情報を用いたフィルタ機構「有り」が最も少ない．特に，go においては,「有り」の割合は，GPP, PBP，PTBP がそれぞれ 10.1%, 9.96%, 9.82%,「無し」が 11.9%，11.7%, 11.6% と 1.7~1.8% 減少している．

5. おわりに

本稿では，パス情報を分岐のフィルタリングに利用できる可能性を示した．3.1 で述べたように，パス長が 4~ 16 あれば，4 ~ 7 割のパスはフィルタリングできる．

また，具体的な実装例としてパス情報を用いたフィルタ機構を提案した．提案手法は，GPP に適用した場合は一定の効果が得られるものの，PBP, PTBP に適用した場合の効果は薄い．今後は，よりフィルタリング率/精度の高い機構の開発が望まれる．

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），および 21 世紀 COE プログラム（課題番号 14213201）による．

参 考 文 献

1) 安藤秀樹: 命令レベル並列処理, コロナ社, chapter 2 (2005).
2) Chang, P. Y., Evers, M. and Patt., Y. N.: Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference, *Proceedings of PACT* (1996).
3) Jiménez, D. and Lin., C.: Dynamic Branch Prediction with Perceptrons, *Proceeding of Seventh International Symposiumon High Performance Computer Architecture*, pp. 197–206 (2001).
4) Jiménez, D. and Lin., C.: Fast Path-Based Neural Branch Prediction, *Proceedings of the 36th Annual International Symposiumon Microarchitecture* (2003).
5) Jiménez, D. and Lin., C.: Piecewise Linear Branch Prediction, *Proceedings of the 32nd International Symposiumon Computer Architecture* (2005).
6) McFarling., S.: Combining branch predictors, Wrl technical report tn-36, Digital Equipment Corporation (1993).
7) 斎藤史子, 山名早人: BTB のエントリ有無を参照した分岐予測器, 先進的計算基盤システムシンポジウム SACSIS2004, pp. 261–268 (2004).
8) 坂和正俊, 田中雅博: ニューロコンピューティング入門, 森北出版株式会社, chapter 1–2 (1997).
9) Sprangle, E. and Carmean., D.: Increasing processor performance by implementing deeper pipelines, *Proceedings of the 30nd International Symposiumon Computer Architecture*, pp. 25–34 (2002).
10) 石井康雄, 平木敬: 実行パス履歴情報を利用した分岐予測手法, 情報処理学会論文誌, Vol. 47, pp. 58–72 (2006).

(平成 13 年 1 月 31 日受付)
(平成 14 年 5 月 10 日採録)