2004 年度 修士論文

予測カウンタの偏向を利用したハイブリッド分岐予測器
～新しい予測表選択手法”Confidece-Selector”の提案～

提出日：2005 年 2 月 2 日
指導：山名早人 助教授

早稲田大学大学院理工学研究科
情報・ネットワーク専攻

学籍番号：3603U099-4

仲沢 由香里

概要

近年のプロセッサは，命令の読み出し，解釈，データの読み出し，実行，実行結果の書き込み等を段階ごとに順次同時に行うパイプライン処理を行っている．しかし，プログラム中に分岐命令が存在すると，分岐先が確定するまで後続の命令を実行できず，パイプラインにストールが発生する．制御依存によるパイプラインストールを緩和するために，近年のプロセッサでは分岐予測が採用されている．分岐予測により，分岐先以降の命令を投機的に実行でき，パイプラインストールを回避できる．しかし，近年，パイプライン段数の深化に伴い，分岐予測ミスペナルティが増大している．そのため，分岐予測ミスの低減は，プロセッサの性能向上のために不可避な問題となっている．

現在まで，様々な分岐予測器が提案されてきた．その中でも，複数の予測器を組み合わせたハイブリッド分岐予測器は，高精度な予測器であることが知られている．ハイブリッド予測器には，各予測器による予測の中から最終的な予測を選択する仕組みが必要となる．多くのハイブリッド分岐予測器は，予測器とは別に Selector を用意し，その Selector により最終的な予測を決定する．しかし，Selector の精度はそれほど高くないという報告もある．また，Selector 用のハードウェアも必要となる．

本研究では，予測器の予測カウンタ状態毎に予測精度が異なることに着目し，予測カウンタ状態に基づいた予測選択手法 Confidence-Selector を提案する．本手法では，従来の Selector が不要となり，Selector に要していたハードウェアを各予測器に割り当てることが可能となる．

SPECint95(train 入力) ベンチマークを対象に，代表的なハイブリッド分岐予測器である Combining 予測器に本手法を適用したところ，12KB のハードウェア容量で平均 0.22%，24KB で平均 0.31%予測ミス率が低減した．フェッチ幅 4 命令の 40 段プロセッサでは，平均 0.31%の予測ミス率低減により，約 4.0%の処理速度 (IPC) 向上を達成できる．

また，パイプライン段数の深化に伴う高クロック化により，予測にかかる時間 (予測遅延) の影響が深刻化している．本手法は Selector が不要なため，これまでのハイブリッド分岐予測器の中では一番シンプルな構成となっており，予測遅延も小さい．そのため，高クロック化が進むにつれ，分岐予測には必要不可欠な手法になると考える．

目 次

第1章　はじめに

近年のプロセッサは，命令の読み出し，解釈，データの読み出し，実行，実行結果の書き込み等を段階ごとに順次同時に行うパイプライン処理を行っている．しかし，プログラム中に分岐命令が存在すると，分岐先が確定するまで後続の命令を実行できず，パイプラインにストールが発生する．この制御依存によるパイプラインストールを緩和するために，多くのプロセッサでは分岐予測が採用されている．分岐予測により，分岐先以降の命令を投機的に実行でき，パイプラインストールを回避できる．しかし近年，パイプライン段数の増加に伴い，分岐予測ミスペナルティが増大している．そのため，分岐予測ミスの低減は，プロセッサの性能向上のために不可避な問題となっている．

分岐予測器は，分岐の成立 (Taken)・不成立 (NotTaken) で遷移する 2bit 飽和カウンタの集まりである予測表 (PHT: Pattern History Table) で構成される．分岐予測器は，1 つの予測表で構成される単体予測器 [1, 2, 3, 4, 5] と，複数の予測表 (単体予測器) で構成されるハイブリッド予測器 [3, 6, 7, 8, 9, 10, 11] に大別される．一般に，ハイブリッド予測器の方が高精度であることが知られている．

ハイブリッド予測器には，各予測器による予測の中から，最終的な予測を選択する仕組みが必要となる．主なハイブリッド予測器は，予測器とは別に Selector を用意し，その Selector により最終的な予測を決定する．予測選択手法には，各予測器の予測成功・失敗の頻度 [3] や，分岐の偏向に基づく [6] 手法がある．しかし，Selector の精度は必ずしも高いわけではない．例えば，代表的なハイブリッド予測器である Combining 予測器 [3] では，SPECint95 ベンチマークで，全予測ミスのうち，選択ミス[1]が 2〜6 割を占める [37].

分岐予測器は有限のハードウェアで如何に予測精度を向上させるかが重要である．ハイブリッド予測器は，複数の予測器と Selector で構成されるため，個々の構成要素に割り当てられる容量は単一の単体予測器と比べて小さくなる．そのため，わざわざ Selector を追加して予測選択を行うのは最善の手法とは言えない．

本研究では，2 つの異なる予測器で構成されるハイブリッド予測器を前提に，一方の予測器の予測カウンタ状態を新たに予測選択の指標に取り入れる．具体的には，一方の予測器の予測カウンタ状態を参照し，予測カウンタが強偏向の場合にはその予測器の予測を採用し，弱偏向の場合にはもう一方の予測器の予測を採用する．1 つの予測器に，予測選択権を与えることで従来の Selector を除去することでき，Selector に要していたハードウェ

[1]正しく選択していればミスをしなかった予測

アを各予測器に再配分することができる．

2 章では，関連研究として動的分岐予測に関する研究をまとめる．3 章では，単体予測器の予測カウンタの各状態における予測精度の解析結果を示す．4 章では，予測カウンタ状態に着目した予測選択手法 Confidence-Selector を提案し，評価，考察を行う．5 章で全体をまとめる．

第2章　分岐予測

分岐命令は，分岐条件成立 (Taken)・不成立 (NotTaken) による分岐方向と，分岐条件が成立した (Taken) 場合の分岐先アドレスが不明なため，後続の命令がフェッチできずにパイプラインの処理が滞る．このような分岐命令による依存関係は，制御依存と呼ばれている．分岐予測は，プログラム中の制御依存を緩和する投機的実行技術である．

分岐予測は，分岐方向予測と分岐先アドレス予測を組み合わせて行う．分岐方向予測は，分岐条件が成立 (Taken) か不成立 (NotTaken) か，つまり，分岐先へジャンプする (Taken) かジャンプしない (NotTaken) かを予測する．分岐方向予測には，プロファイルやコンパイルによってあらかじめ予測を行う静的予測と，実行中に分岐の過去の振る舞いから予測を行う動的予測がある．近年，多くのプロセッサに実装されている分岐方向予測は動的予測であり，本研究でも動的分岐方向予測の精度を向上することを目標としている．

動的分岐予測器 (以下，分岐予測器と呼ぶ) は，分岐条件の成立 (Taken)・不成立 (NotTaken) で遷移する 2bit 飽和カウンタ (図 2.1) の集まりである予測表 (PHT:Pattern History Table) で構成される．本研究では，2bit 飽和カウンタを予測カウンタと呼ぶ．分岐予測器は，この予測カウンタの状態を参照して分岐方向を予測する．これは，過去の分岐方向の傾向と将来の分岐方向の間に相関があることを利用している．予測表は，分岐命令アドレスや分岐履歴などでインデクス付けられ，様々なインデクス方法が提案されている．

一方，分岐先アドレス予測は，分岐条件が成立 (Taken) した場合の分岐先アドレスを予測する．分岐方向予測により分岐条件が成立する (Taken) と正しく予測できても，分岐先アドレスがわからないと後続の命令がフェッチできずパイプラインはストールする．そのため，分岐方向だけでなく，分岐先アドレスの予測も必要とされている．分岐先アドレス予測は，分岐先アドレスバッファ(BTB:Branch Target Buffer)[12] で実現される．BTB は，分岐条件が成立 (Taken) した分岐のみをエントリに登録し，分岐先アドレスを記憶する．

本章では，動的分岐方向予測の関連研究についてまとめる．現在までに，様々な分岐予測器が提案されている．分岐予測器は，単一の予測表で構成される単体予測器と，複数の予測表で構成されるハイブリッド予測器に分類できる．また，従来の分岐予測器を対象に予測精度向上を試みる研究も行われている．

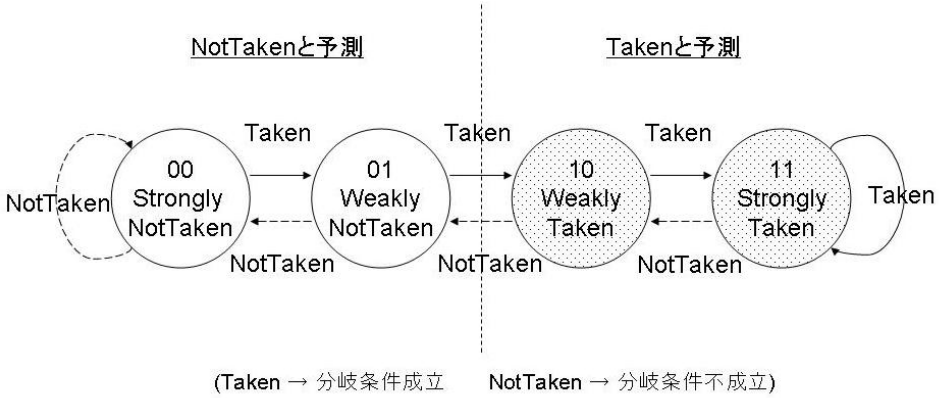


図 2.1: 2bit 飽和カウンタの状態遷移図

2.1 単体予測器

単体予測器は，１つの予測表 (PHT:Pattern History Table) で構成される．本節では，Bimodal 予測器 [1]，2 レベル適応型予測器 [2]，gshare 予測器 [3]，Alloyed 予測器 [4]，Agree 予測器 [5] について説明する．Bimodal，2 レベル適応型，gshare 予測器は，他研究での比較対象や，ハイブリッド予測器の構成要素としてもよく使われる，代表的な単体予測器である．また，Alloyed 予測器は予測表のインデクス方法に，Agree 予測器は予測カウンタの遷移方法に工夫を凝らした単体予測器である．

Bimodal 予測器 [1]

Bimodal 予測器は，1981 年に Smith によって考案された最初の分岐予測器である．Bimodal 予測器は，分岐命令アドレスの下位ビットをインデクスとした予測表 (PHT) で構成されるシンプルな予測器である (図 2.2)．

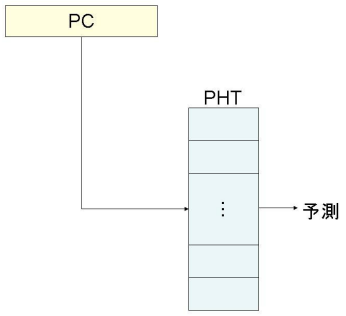


図 2.2: bimodal 予測器 [1]

2 レベル適応型分岐予測器 [2]

2 レベル適応型予測器 (2-level Adaptive Branch Predictor) は，現在提案されている分岐予測器の基礎となる予測器で，1993 年に Yeh らによって提案された．2 レベル適応型予測器は 2 階層の表で構成される．1 階層目の表 (BHR, BHT: Branch History Register or Table)[1]には，分岐条件が成立する (Taken) 場合には 1，成立しない (NotTaken) 場合には 0 として過去数回の実行結果 (分岐履歴) を格納する．2 階層目の表は，1 階層目の表に格納された分岐履歴でインデクス付けられた予測表 (PHT) で構成される．Yeh らは，BHR(BHT) と PHT において，global，per-address，per-set の各 3 種類のインデクス方法を考案し，その組み合わせから 9 種類の予測器 (表 2.1) を提案している [2]．各予測器の構成を図 2.3 に示す.

- global
  予測対象となる分岐命令以前に実行された有限個の分岐命令の履歴を記録する．

- per-address
  静的分岐毎に履歴を記録する．静的分岐毎にエントリが取れない場合は，インデクスに用いられない分岐アドレスの上位ビットをタグとして利用し，タグの不一致によってエントリを入れ替える[2]．

- per-set
  グループ (分岐のオペコード，分岐クラス，分岐アドレス等で決定される) 毎に履歴を記録する．

また，GAg 予測器のインデクスを，グローバルな分岐履歴と分岐命令アドレスの排他的論理和とした gshare 予測器 [3] も提案されている (図 2.4)．gshare はシンプルな構成で，かつ高精度の予測正解率を達成することから，多くの研究で用いられている．

表 2.1: 2 レベル適応型予測器の種類 [2]

BHT\PHT	global	per-address	per-set
glboal	GAg	GAp	GAs
per-address	PAg	PAp	PAs
per-set	SAg	SAp	SAs

[1]Register は単一の履歴，Table は複数の履歴を表す.

[2]複数の PHT の選択では，分岐アドレスの下位ビットによる選択を per-address と呼ぶ場合がある．本稿では，Yeh らの分類 [2] を尊重し，下位ビットによる選択を per-set と分類する.

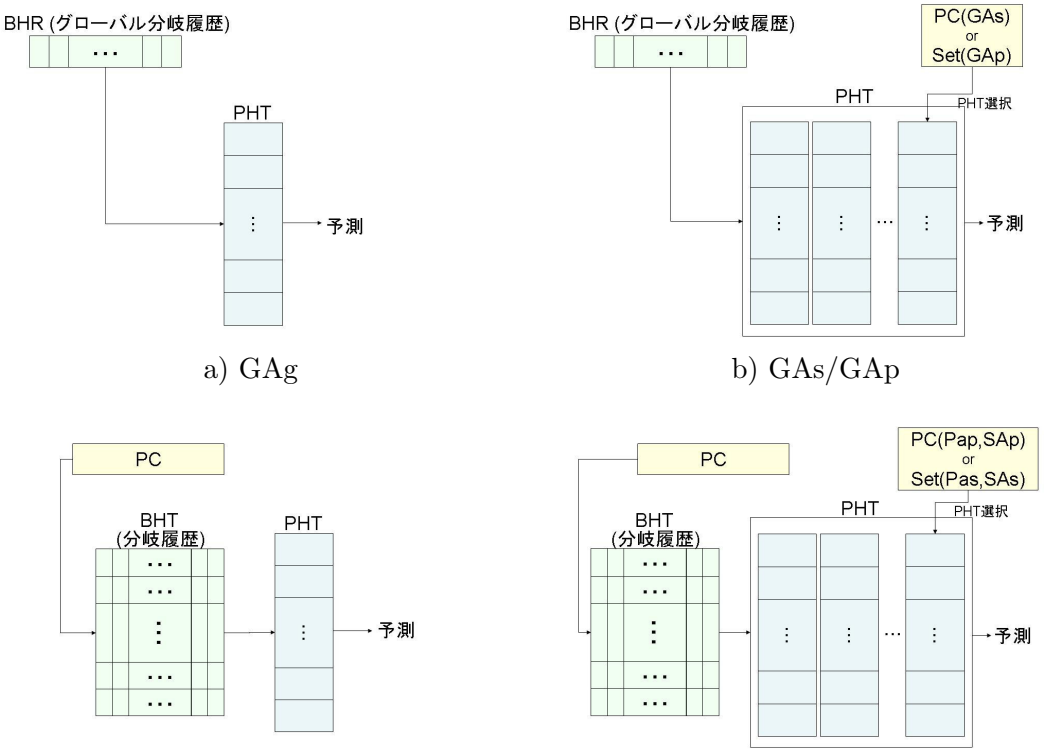


図 2.3: 2 レベル適応型予測器 [2]

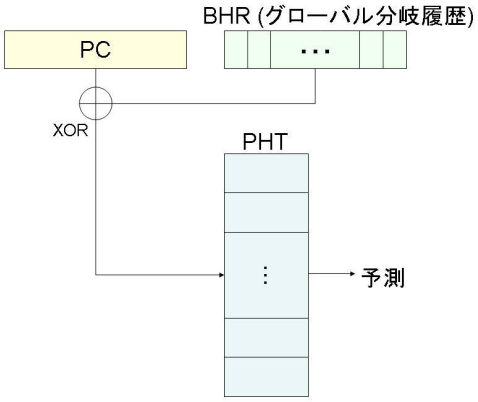


図 2.4: gshare 予測器 [3]

Alloyed 予測器 [4]

分岐の履歴には，Global 分岐履歴と Local 分岐履歴がある．

- Global 分岐履歴
  プログラムの実行順の分岐条件の成立 (Taken)・不成立 (NotTaken) の履歴
- Local 分岐履歴
  分岐命令毎の分岐条件の成立 (Taken)・不成立 (NotTaken) の履歴

Global 履歴と Local 履歴では，予測可能な分岐パターンが異なる．そこで，2000 年，Skadron らは，分岐アドレスの下位ビット，Global 履歴，Local 履歴の連結を予測表 (PHT) のインデクスに利用した Alloyed 予測器を提案した [4](図 2.5)．彼らは，シミュレータに SimpleScalar 2.0/PISA[23] を複数パスの実行を可能にした HydroScalar，ベンチマークに SPECint95，IBS[29] から gnuchess，Smith の Unix-Utils[30] から wolf，SPLASH2[31] から radiosity, volrend を採用して実験を行った．Cycle-Level シミュレーションは 1 億命令，Instruction-Level シミュレーションは 10 億命令を対象にした．その結果，Alloyed 予測器は，8KB GAs 予測器の予測ミス率のうち平均 23.1%削減した．

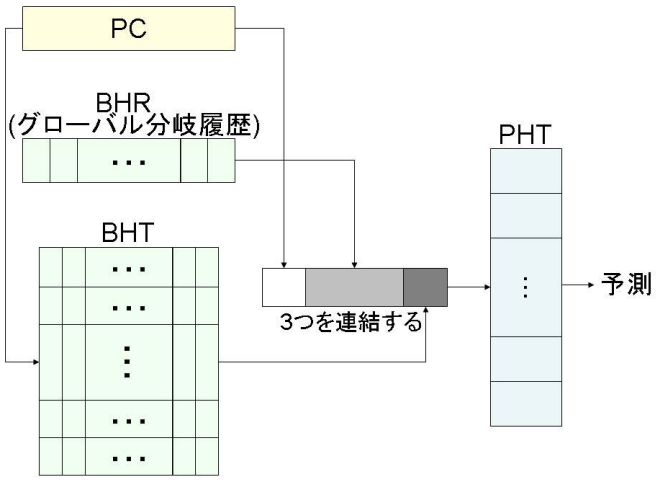


図 2.5: Alloyed 予測器 [4]

Agree 予測器 [5]

1997 年，Sprangle らは，BTB(Branch Target Buffer) の各エントリに Direction Bit を追加することで分岐の偏向を記憶する Agree 予測器を提案した [5]．Direction Bit は，分岐命令が BTB に格納された時点の分岐結果 (Taken/NotTaken) を記録する．Agree 予測器は，

予測表の 2bit 飽和カウンタの遷移方法が他の分岐予測器と異なる．分岐方向と Direction Bit が一致 (Agree) した時はインクリメントし，不一致 (Disagree) の時にはデクリメントする (図 2.6)．予測は，2bit 飽和カウンタの状態を参照することで，Direction Bit と一致する方向に分岐するか否かを決定する．

Agree 予測器は，分岐命令の多くが”Taken”あるいは”NotTaken”のどちらかに偏っていることを利用し，予測表の破壊的競合の緩和を試みた予測器である．破壊的競合とは，予測表の同一エントリを異なる分岐偏向をもつ分岐が利用することによって予測ミスが起きる現象である．分岐偏向は，分岐条件の成立 (Taken)・不成立 (NotTaken) の割合を意味する．つまり，破壊的競合とは，”Taken”が多い分岐と”NotTaken”が多い分岐が同一エントリを使用することで，2bit 飽和カウンタが逆偏向に更新され，正しく予測ができなくなる現象である．Agree 予測器では，初めに設定する Direction Bit が正しく分岐の偏向を捕らえられればいいが，Direction Bit が分岐の偏向と異なると”Disagree”が多くなる．”Agree”が多い分岐と”Disagree”が多い分岐が同一のエントリを使用する状況が発生すれば，破壊的競合が生じる．

Sprangle らは，シミュレータに Sun 4mWorkstation をモデル化した SoSS，ベンチマークに SPECint95 を採用して実験を行った．各々のプログラムは gcc -O3 でコンパイルされており，また，4K エントリ direct-mapped BTB を採用している．実験の結果，Agree 予測器は，gshare 予測器よりも常に高い予測正解率に達することを示した．また，小容量の方が，Direction bit 参照による予測カウンタの遷移方向調整の効果が大きく，1K エントリの予測表で，gshare 予測器よりも最大約 8%(vortex) 高い予測正解率に達している．

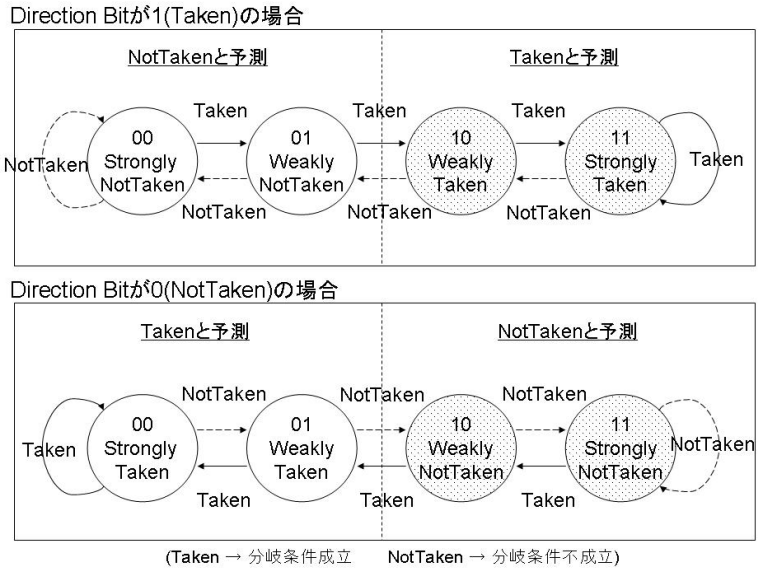


図 2.6: Agree 予測器 [5] の 2bit 飽和カウンタの動作

2.2 ハイブリッド予測器

ハイブリッド予測器は複数の予測器 (単体予測器/ハイブリッド予測器) で構成され，各予測器による予測の中から最終的な予測を選択する．現在までに，様々な構成のハイブリッド予測器が提案されてきた [3, 6, 7, 8, 9, 10, 11]．本節では，予測選択手法に着目し，予測器の信頼度，分岐偏向，多数決，オペランド値の 4 種類に分類して，各ハイブリッド予測器を説明する．

2.2.1 予測器の信頼度による予測選択

各予測器の予測信頼度を元に予測を選択する手法として，Combining 予測器 [3] がある．

Comgining 予測器 [3]

Combining 予測器は，1993 年，McFarling らによって，初のハイブリッド予測器として提案された [3]．Combining 予測器は 2 つの単体予測器と 1 つの Selector(選択器) で構成される (図 2.7)．Selector は予測の成功・失敗で遷移する 2bit 飽和カウンタのテーブルで，カウンタの値によって予測を採用する単体予測器を決定する．予測の採用・不採用に関係なく，常に両方の単体予測器を更新する．Combining 予測器は，構成要素となる 2 つの単体予測器が，異なる分岐パターンを予測できるほど精度が向上する．

本研究では，SAg 予測器 [2] と gshare 予測器 [3] を組み合わせた Combining 予測器を利用する (第 4 章参照)．

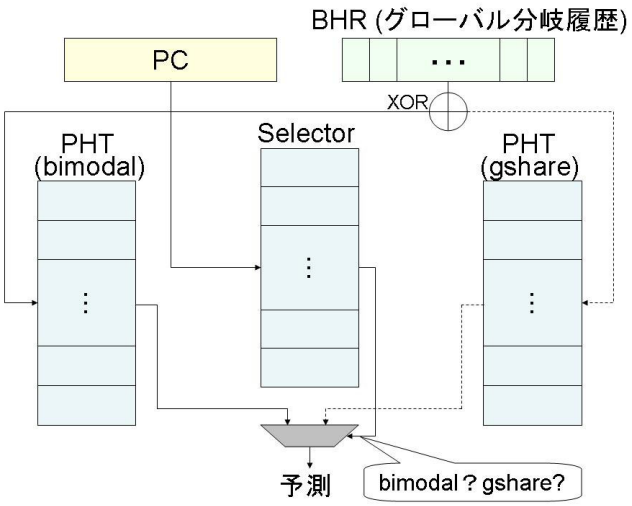


図 2.7: Combining 予測器 [3](bimodal 予測器 [1] と gshare 予測器 [3] の組み合わせ)

2.2.2 分岐の偏向による予測選択

分岐の偏向を利用して最終予測を決定するハイブリッド予測器には，Bi-Mode 予測器 [6]，YAGS(Yet Another Global Scheme) 予測器 [7] がある．

Bi-Mode 予測器 [6]

Bi-Mode 予測器は，1997 年に Lee らによって，gshare 予測器の破壊的競合 (前節の Agree 予測器参照) を緩和することを目的として提案された．Bi-Mode 予測器は，ChoicePHT，TakenPHT，NotTakenPHT の 3 つの予測表 (PHT) から構成される．ChoicePHT は Bi-modal 予測器 [1]，TakenPHT と NotTakenPHT(合わせて DirectionPHT と呼ぶ) は gshare 予測器 [3] で構成される．ChoicePHT が”Taken”と予測する場合は，TakenPHT における予測が採用され，ChoicePHT が”NotTaken”と予測する場合は，NotTakenPHT における予測が採用される．”Taken”が多い分岐は TakenPHT を頻繁に利用し，”NotTaken”が多い分岐は NotTakenPHT を頻繁に利用するため，PHT における破壊的競合を緩和することができる．ChoicePHT は常に更新されるが，DirectionPHT は予測が採用された PHT のみ更新される．

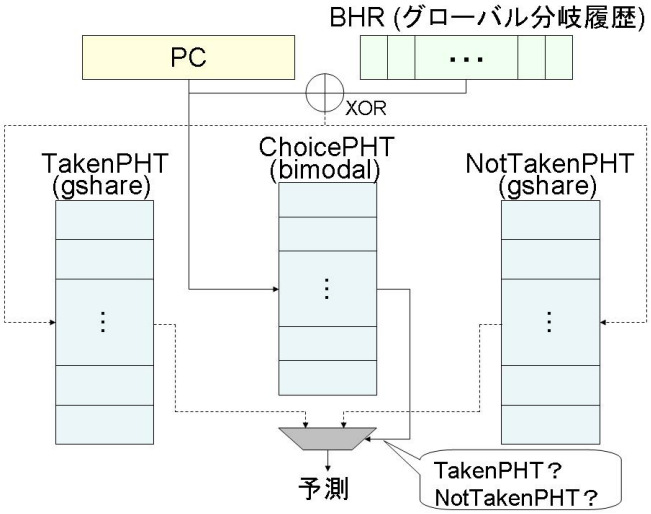


図 2.8: Bi-Mode 予測器 [6]

2003 年吉瀬らは，Bi-Mode 予測器に改良を加えた Bimode-Plus 予測器 [15, 16] を提案した．Bimode-Plus 予測器は，プログラム実行中，常に分岐条件が成立する (Taken) 分岐，あるいは常に成立しない (NotTaken) 分岐が多数存在することに着目し，これらの分岐を常に”Taken”あるいは”NotTaken”と予測する．Bimode-Plus 予測器は，Bi-Mode 予測器の ChoicePHT を構成する 2bit 飽和カウンタを，taken_flag と untaken_flag という 2 つの

フラグとして兼用する．2bit 飽和カウンタを taken_flag，untaken_flag として利用している状態を Plus モードと呼び，通常の飽和カウンタとして利用している状態を 2BC モードと呼ぶ．Plus モードと 2BC モードを区別するために，ChoicePHT の各エントリに 1bit のフラグ (valid ビット) を追加する．

valid ビットを追加した 2bit カウンタの状態遷移を図 2.9 に示す．下の枠中 (2BC モード) の 4 つの状態が通常の 2bit 飽和カウンタで，上の枠中 (Plus モード) に示す 3 つの状態が追加された状態である．初期状態は上枠中央の valid ビットを含む 3 ビットが全て 0 の状態である．分岐条件が常に成立 (Taken) の場合，カウンタは”All Taken”の状態に留まる．一方，分岐条件が常に不成立 (NotTaken) の場合には，”All NotTaken”の状態に留まる．valid ビットが 0 で Plus モードの時には Choice PHT の値から予測方向を決定する．そうでない (2BC モード) 場合には，Bimode 予測器と同様に Direction PHT の予測を採用する．有効ビットが 0 で Plus モードの時には，Direction PHT は更新しない．有効ビットが 1 で 2BC モードの時には Bimode 予測器と同様に，予測を採用した Direction PHT を更新する．

吉瀬らは，シミュレータに SimAlpha[32]，ベンチマークに SPECint95 を採用して実験を行った．12KB の Bi-Mode 予測器と比較して，14KB の Bi-ModePlus 予測器は予測ミス率が平均 0.19%低減した．

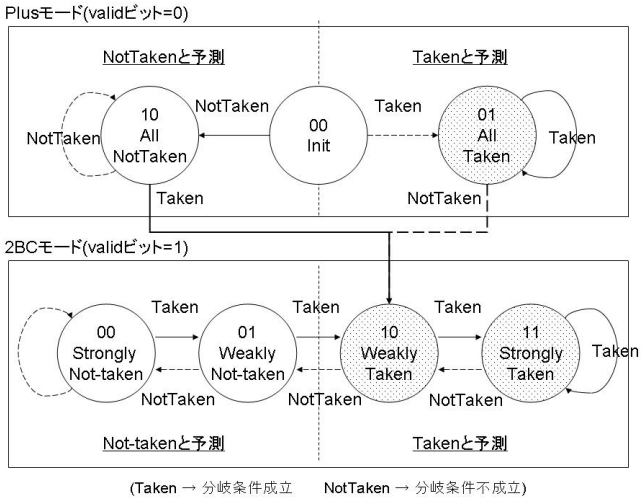


図 2.9: BimodePlus 予測器 [?] の 2bit 飽和カウンタの動作

YAGS 予測器 [7]

Bi-Mode 予測器では ChoicePHT によって分岐の偏向を判断し，予測を採用する DirectionPHT を選択する．しかし，ChoicePHT による分岐偏向の判断が実際の結果と等しけ

れば，わざわざ DirectionPHT で予測を行う必要がない．そこで，1998 年 Eden らは，予測の冗長性を排除するために，Bi-Mode 予測器と異なる予測表選択アルゴリズムを採用した YAGS(Yet Another Global Scheme) 予測器を提案した [7]．YAGS 予測器は，Bi-Mode 予測器と同様に 3 つの PHT から構成されるが，Bi-Mode 予測器とは異なり DirectionPHT に分岐を区別するためのタグを用意する．YAGS 予測器は，次の手順で予測を行う．

1. ChoicePHT が”Taken”と示した場合には，NotTakenPHT を参照し，当該分岐のエントリの有無をタグの一致・不一致で確認する (ChoicePHT が”NotTaken”と示した場合には，TakenPHT を参照し，同じくエントリの有無を確認する).

2. NotTakenPHT にエントリがあった場合には，NotTakenPHT の予測を採用する．エントリがない場合には，ChoicePHT の予測 (つまり”Taken”) を採用する．逆も同じである．

Eden らは，ベンチマークに SPEC95(kernel trace を含む．6 万命令毎に Context Switch) を採用して実験を行った．0.1〜100KB の予測器容量において，6bit のタグの YAGS 分岐予測器は，Bi-Mode 予測器 [6]，Gskewed 予測器 [8]，ghsare 予測器 [3] と比較しても常に高精度であることを示した．しかし，文献 [26] では，YAGS 分岐予測器を SimpleScalar[23] に実装し，SPECint95(ref 入力) を対象に実験を行った結果，Bi-Mode 分岐予測器 [6] や Combining 分岐予測器 [3] よりも，YAGS 分岐予測器の方が予測ミス率が高いと報告している．これは，YAGS 分岐予測器に追加されたタグが予測器容量を大きくしてしまったためと考えられる．

2.2.3 全予測を利用した予測選択

第 2.2.1 節と第 2.2.2 節に分類されるハイブリッド予測器は，全ての予測器の予測結果が最終予測に反映されているわけではない．全ての予測器の予測を参照して，最終的な予測を決定するハイブリッド予測器も提案されている．代表的な予測器に，多数決で予測を決定する Skewed 予測器 [8]，過去の予測精度による重み付きの多数決で決定する WMBP(Weighted Majority Branch Predictor)[9] 等が挙げられる．また，WMBP を基に，複数の予測表の予測をインデクスに利用して，最終予測を決定する予測カウンタを選択する COLT(Combined Output Lookup Table) 予測器 [10] も提案されている．ここでは，Skewed 予測器と COLT 予測器について説明する．

Skewed 予測器 [8]

Skewed 予測器は，1997 年に Michaud らによって提案された．Skewed 予測器は，3 つの予測表 (PHT) で構成される (図 2.10)．3 つの予測表は，分岐アドレスとグローバル分

岐履歴を用いて，各々異なるマッピング関数によってインデクス付けられる．各予測表の予測の多数決をとることで最終的な予測を決定する．3 つのマッピング関数は，1 つの予測表で競合が発生する場合は，他の 2 つの予測表では競合が発生しないように設定する．Michaud らは，3 つの予測表全てを，分岐アドレスとグローバル分岐履歴を用いてインデクス付ける Gskewed 予測器と，3 つの予測表のうち 1 つだけは，分岐アドレスのみでインデクス付ける Enhanced Gskewed(Egskewed) 予測器を提案している．

彼らは，トレース駆動型シミュレータで，IBS ベンチマーク [29] を対象に実験を行った．その結果，Egskewed 予測器は，gshare 予測器 [3] の約半分のハードウェア量で同等の予測正解率に達成することを示した．また，予測表は 3 つを組み合わせれば充分であること，全ての予測表を更新するよりも，正しい予測を導いた予測表のみを更新する方が精度がよいことを示した．

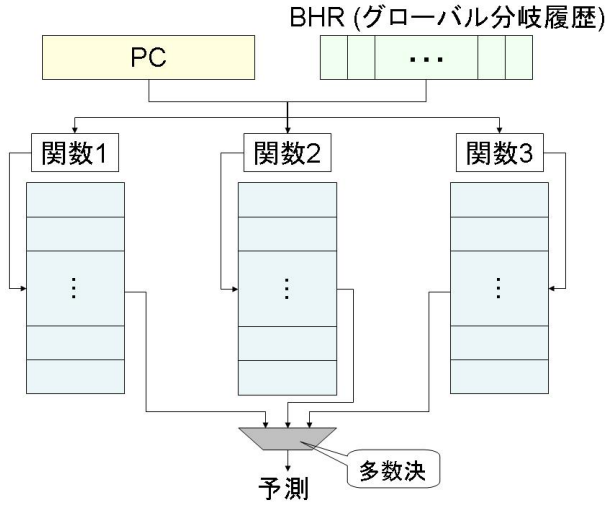


図 2.10: Skewed 予測器 [8]

COLT 予測器 [10]

COLT 予測器は，2002 年に Loh らによって提案された．COLT 予測器は，各予測器の予測をインデクスに利用することで，最終的な予測カウンタを選択する (図 2.11)．COLT 予測器は，分岐履歴 (h bit)，分岐命令アドレス (a bit) と，各予測器の予測 (n bit) の連結でインデクス付けられた Vector of Mapping Tables(VMT) の c-bit 飽和カウンタで最終予測を行う．VMT の飽和カウンタは，従来の予測表 (PHT) と同様に分岐条件が成立 (Taken) の場合にはインクリメント，不成立 (NotTaken) の場合にはデクリメントする．

彼らは，シミュレータに SimpleScalar 4.0 MASE/Alpha[24]，ベンチマークに SPECint2000(1 億分岐命令スキップ後の 5 億分岐命令) を採用して実験を行った．COLT 予測器の予測表

には，Bimodal 予測器 [1]，gshare 予測器 [3]，Bi-Mode 予測器 [6]，Egskewed 予測器 [8] など 11 個の予測器から最適な組み合わせが選ばれた．その結果，16〜256KB の COLT 予測器は，同容量の Alloyed 履歴パーセプトロン予測器[3]よりも約 0.3〜0.4%予測ミス率を低減した．COLT 予測器は，複数の予測表で構成される複雑な予測器であるため，予測が必要な場合に予測が間に合わない，予測遅延が問題となる．予測遅延については，COLT 予測器を構成する予測表に様々な容量を用意することで，緩和可能と考えられる．つまり，予測遅延が生じた場合には，早期に予測結果が出る小容量の予測表の予測を採用することで解決できる．

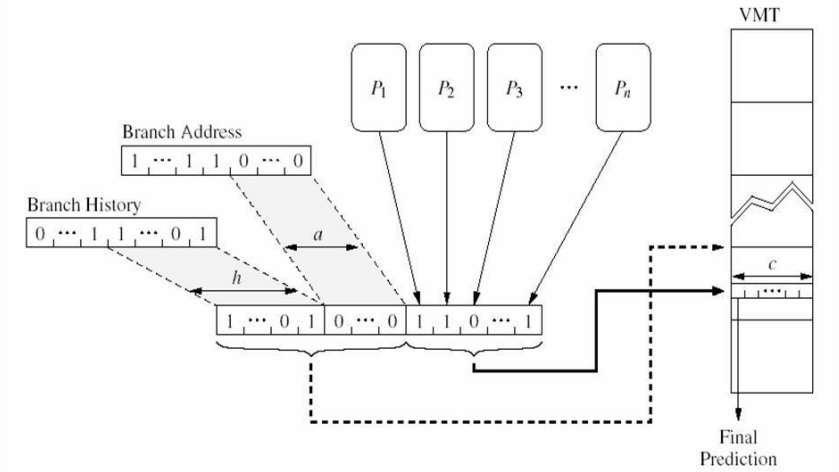


図 2.11: COLT 予測器 (文献 [10]Fig.3 より引用)

2.2.4 オペランド値を利用した予測選択

ソースオペランド値を利用した予測選択手法として，Branch Difference 予測器 [11] がある．

Branch Difference 予測器 [11]

BDP 予測器は，1999 年に Heil らによって提案された．BDP 予測器は，分岐のソースオペランド値の差分を利用して予測を選択する．BDP 予測器は，通常の予測を行う Backing 分岐予測器，Backing 分岐予測器で予測困難な分岐のみ予測する Rare Event 予測器 (REP) と，ソース・オペランド値の差分の履歴を格納する表 (VHT: Value History Table) で構成される (図 2.12)．通常は Backing 分岐予測器で予測を行うが，当該分岐命令の Value

[3]PHT に飽和カウンタではなく，パーセプトロンを用いた手法 [38, 28]．予測を導くための計算時間がかかるため，予測遅延が大きい．

History が，Rare Event 予測器のタグと一致する場合だけ，REP 予測器の予測を採用する．Rare Event 予測器は，Backing 分岐予測器で予測を行い，かつ，その予測がミスした時に更新される．Backing 分岐予測器は，予測が採用された時だけ更新される．

Heil らは，シミュレータに SimpleScalar2.0[23]，ベンチマークに SPECint95[4]を採用して実験を行った．Backing 予測器に Bi-Mode 予測器を採用した場合，単一の Bi-Mode 予測器と比較して，16〜128KB で平均約 1%予測ミス率が低減した．

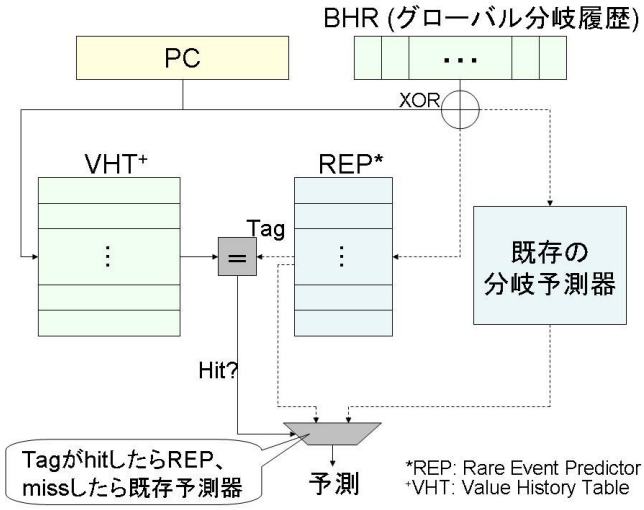


図 2.12: BDP 予測器 [11]

2.3 単体/ハイブリッド予測器の予測精度向上手法

第 2.1,2.2 節では，これまでに提案された分岐予測器を単体予測器とハイブリッド予測器に分類して紹介した．これらの分岐予測器の予測精度を向上させるための研究も行われている．本節では，分岐の偏向を利用して予測表における競合を緩和する手法 [13, 14, ?] と，信頼度判定器を導入して予測の信頼度を測る手法 [17, 18, 19, 20] を説明する．

2.3.1 分岐の偏向を利用した競合緩和手法

分岐命令の多くは，偏向が強い (”Taken”や”NotTaken”の一方が極端に多い) と言われている．文献 [21] では，SPECint95 ベンチマークにおいて，全実行回数の 90%以上が全て”Taken”，あるいは全て”NotTaken”である静的分岐命令が，全静的分岐命令の約 63%を占めると報告している．つまり，半数以上の分岐は，”Taken”あるいは”NotTaken”のどち

[4]compress, gcc, go, ijpeg, li

らかに偏っている．そこで，偏向の強い分岐は，直接その偏向に従って予測を行い，それ以外の分岐だけを通常の分岐予測器で予測することで，予測精度の向上を試みる手法が提案されている [13, 14, ?]．偏向の強い分岐を通常の分岐予測器のエントリからはずすことで，予測表における競合を緩和することができる．特に [13] と [14] では，分岐の偏向を判断するために，分岐先アドレスバッファ(BTB: Branch Target Buffer) を利用している．BTB は分岐先アドレス予測に利用されており，分岐条件が成立する (Taken) 分岐の分岐先アドレスをエントリに登録する．

Filtering 機構 [13]

Filtering 機構は，1996 年に Chang らによって提案された．Filtering 機構は，BTB の各エントリに Direction Bit と飽和カウンタを追加する．最初に分岐結果が得られた時に，その分岐結果を Direction Bit に格納し，カウンタを 0 に初期化する．その後，分岐結果と Direction Bit が等しい場合にはカウンタをインクリメントし，異なる場合には，Direction Bit を反転して，カウンタを 0 に初期化する．カウンタが飽和している場合にだけ，当該分岐命令を偏向した分岐命令を判断して，Direction Bit の値を予測値として利用する．それ以外の時は，2 レベル適応型予測器 [2] による予測を採用する．2 レベル適応型予測器は，予測が採用された時だけ更新される．

Chang らは，トレース駆動型命令レベルシミュレータで，SPECint95 ベンチマークを対象に実験を行った．その結果，gshare 予測器 [3] に適用した場合，2〜16KB で，gshare 予測器よりも平均 0.02%予測ミス率が低減した．

BTB 参照 [14]

2004 年，斎藤らは，BTB にエントリを持たない分岐命令の多くが，分岐条件が成立しない (NotTaken) 分岐であることから，BTB にエントリを持たない分岐を常に”NotTaken”と予測する手法を提案した．この手法では，BTB にエントリを持つ分岐のみ，通常の分岐予測器で予測を行う．また，通常の分岐予測器は，予測が採用された時だけ更新される．シミュレータに SimpleScalar 3.0c/PISA[23]，ベンチマークに SPECint95(train 入力) を採用して実験を行った．その結果，8KB の gshare 予測器 [3] で平均 1.5%，1.5KB の Bi-Mode 予測器 [6] で平均 0.4%，予測ミス率が低減した．

2.3.2 信頼度判定の導入

予測の信頼度を測り，信頼度が低い場合に予測結果を反転する手法として，飽和カウンタを用いた信頼度判定手法 [17] とソースオペランド値を利用した BPRU 予測器 [18, 19, 20]

を説明する．

飽和カウンタを用いた信頼度判定 [17]

1999 年，Manne らは，2bit 飽和カウンタを用いて予測の信頼度を判定し，信頼度が低い場合に予測結果を反転する SBI(Selective Branch Inversion) 予測器を提案した (図 2.13)．信頼度の判定には，2bit 飽和カウンタのテーブルである信頼度判定器を用いる．2bit 飽和カウンタは，予測が成功した場合インクリメントされ，失敗した場合 0 に初期化される．信頼度判定器は，分岐命令アドレスの下位ビットと Global 分岐履歴の XOR でインデクスされる．

Manne らは，シミュレータに SimpleScalar[23]，ベンチマークに SPECint95 を採用して実験を行った．Combining 予測器 [3]，Bi-Mode 予測器 [6] に対して信頼度判定器を適用した結果，最大 0.5%程度予測ミス率が低減した．

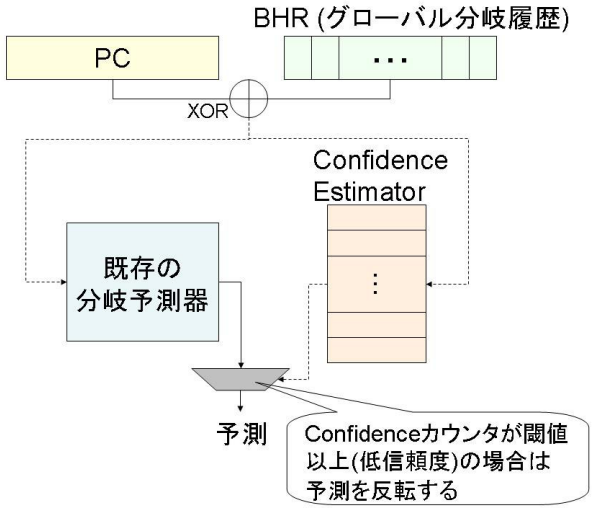


図 2.13: SBI 予測器 [17]

ソースオペランド値の利用 [18, 19, 20]

2001 年，Juan らは，分岐命令のソースオペランド値予測を利用して予測の信頼度を評価し，信頼度の低い予測を反転する BPRU(Branch Prediction Reversal Unit) 予測器を提案した．信頼度判定器 (Reversal Table) は，従来の分岐予測器による予測結果，Global 分岐履歴，ソースオペラント値を生成する命令のアドレス，予測したオペランド値でマッピングした 2bit 飽和カウンタのテーブルで構成される (図 2.14)．信頼度判定器には各エントリは分岐命令アドレスの一部を Tag として保持している．信頼度判定器のカウンタは，

予測が失敗するとインクリメントされ，成功するとデクリメントされる．カウンタ値が閾値以上の場合に，予測の信頼度が低いと判断し，従来の分岐予測器による予測を反転した結果を最終的な予測として採用する．

Juan らは，シミュレータに SimpleScalar 3.0c/Alpha[23]，ベンチマークに SPECint2000 を採用して実験を行った．Alpha 21264[32] の Combining 分岐予測器に BPRU を適用したところ，容量が大きいほど予測ミス率が低減し，128KB で平均 1%予測ミス率が低減した．

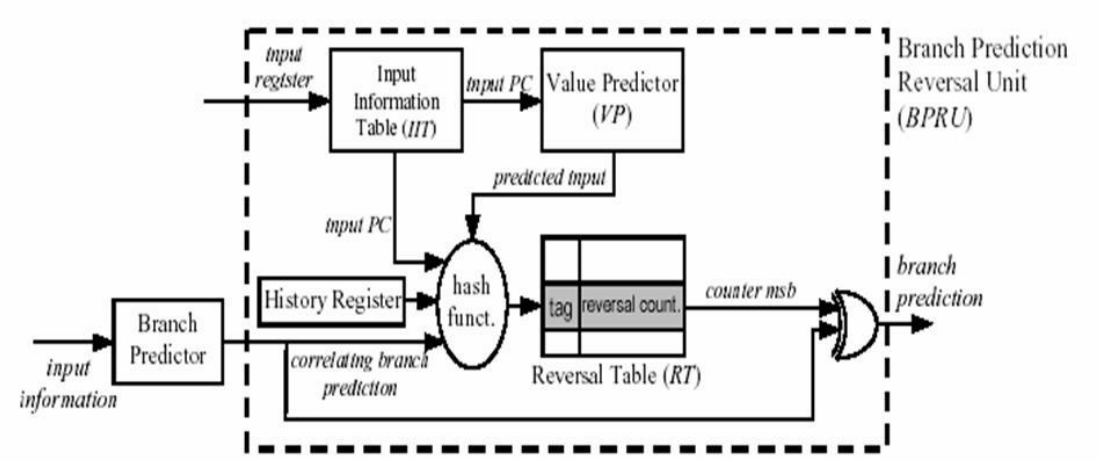


図 2.14: BPRU 予測器 (文献 [20] より引用)

2.4 関連研究のまとめ

本章では，動的分岐予測器の関連研究を，単体予測器，ハイブリッド予測器，単体/ハイブリッド予測器の予測精度向上手法の 3 つに分類して紹介した．各研究を表 2.2-2.4 にまとめる．各研究毎に，予測器 (予測手法) 名，提案者，提案年，概要，特徴，評価条件，評価結果を示す.

表 2.2:　単体予測器

予測器	提案者	年	概要	特徴	実験環境 [1]	評価結果
Bimodal[1]	Smith (ControlData 社)	1981	分岐アドレスの下位ビットでインデックス	初めての分岐予測器	科学計算系プログラム [2]	16 エントリで約 80-99.5%の予測正解率
2 レベル適応型 [2]	Yeh (Michigan 大)	1993	ローカル or グローバル履歴でインデクス 9 種類のインデクス方法を提案 (GAg, GAs, GAp, PAg, PAs, PAp, SAg, SAs, SAp)	過去の履歴と分岐結果に相関があることを利用	トレース駆動型 SPEC89[3](int 系は 2 千万条件分岐命令，fp 系は 1 億命令まで)	GAs, PAs, SAs の中で, 1KB では PAs (平均 96.3%), 16KB では GAs (平均 97.2%) が一番正解率が高い．しかし，各予測器の正解率の差は約 1%程度しかない.
gshare[3]	McFarling (WRL)	1993	分岐アドレスとグローバル履歴の XOR でインデックス	GAg(2 レベル適応型) の変形版	SPEC98 (初めの 1000 万命令)	1KB で，Bimodal より平均約 2.5%予測正解率が高い
Alloyed[4]	Skadron (Virginia 大)	2000	分岐アドレス，PC 毎ローカル履歴，グローバル履歴の連結でインデックス		SimpleScalar2.0/PISA SPECint95/IBS/Unix-Utils/SPLASH2[4]	8KB で，GAs 予測器の予測ミスを平均 23.1%削減
Agree[5]	Sprangle (Michigan 大)	1997	BTB の各エントリに Direction Bit を追加 Direction Bit の値と一致するかで予測カウンタを遷移	分岐の偏向を利用して予測表の破壊的競合を緩和	SoSS[5] SPECint95	1K エントリで，gshare 予測器より約 1〜8%予測正解率が高い (小容量の方が向上率が大きい)

[1] 使用したシミュレータとベンチマーク

[2]ADVAN, SCI2, SINCOS, SORTST, GIBSON, TBLLNK

[3](int)eqntott, espresso, gcc, li (fp)doduc, fpppp, matrix300, spice2g6, tomcatv

[4]IBS は gnuchess, Unix-Utils は wolf, SPLASH2 は radiosity, volrend を採用

[5]Sun 4mWorkstation をモデル化したシミュレータ

表 2.3:　ハイブリッド予測器

予測器	提案者	年	概要	特徴	実験環境	評価結果
Combining[3]	McFarling (WRL)	1993	2 つの単体予測器と 1 つの Selector で構成 Selector は予測の成功・失敗で遷移する 2bit 飽和カウンタの表	構成要素となる 2 つの単体予測器が，異なる分岐パターンを予測できるほど精度 UP	SPEC98 (初めの 1000 万命令)	Bimodal と gshare を組み合わせた場合，1KB で単一の gshare より平均約 1.0%予測正解率が高い (4KB の gshare と同等)
Bi-Mode[6]	Lee (Michigan 大)	1997	3 つの予測表で構成 1 つの予測表で分岐の偏向を判断し，残り 2 つの予測表のどちらを参照するか決定する	gshare 予測器における破壊的競合を緩和	トレース駆動型 SPECint95/IBS-Ultrix	6KB の Bi-Mode は，16KB の gshare と同等の予測正解率を達成
Bimode-Plus[?]	吉瀬 (電通大)	2003	常時 Taken，常時 NotTaken の分岐は，分岐の偏向をそのまま予測とする	Bi-Mode 予測器における競合を緩和	SimAlpha SPECint95	14KB の場合，12KB の Bi-Mode 予測器より平均 0.19%予測ミス率が低い
YAGS[7]	Eden (Michigan 大)	1998	Bi-Mode 予測器の DirectionPHT にタグを追加	Bi-Mode 予測器の予測の冗長性を排除	SPEC95 (kernel trace 含)	0.1〜100KB において，Bi-Mode, gskewed, gshare 予測器より常に高精度
Skewed[8]	Michaud (irisa 大)	1997	異なるマッピング関数でインデクスされた 3 つの予測表で構成	マッピング関数は，2 つの予測表で競合が発生しないように設定	トレース駆動型 IBS-Ultrix	gshare 予測器の約半分の容量で同等の予測正解率を達成
COLT[10]	Loh (Yale 大)	2002	複数の予測器と，各予測器の予測の連結でインデクスされた n ビット飽和カウンタの表で構成	全ての予測を利用	SimpleScalar4.0MASE SPECint2000(1 億命令スキップ後の 5 億分岐命令)	16〜256KB において，Alloyed 履歴パーセプトロン予測器より約 0.3-0.4%予測ミス率が低い　gshare や Bi-Mode 等の代表的な予測器とは比較されていない
BDP[11]	Heil (Wisconsin 大)	1999	既存分岐予測器，予測困難な分岐を予測する予測器 (REP)，ソースオペランド値の差分を格納する表で構成 ValueHistory が REP のタグと一致する時のみ REP の予測を採用	分岐のソースオペランド値の差分を利用	SimpleScalar2.0 SPECint95[1]	16〜128KB において，Bi-Mode 予測器より平均約 1%予測ミス率が低い

[1]compress, gcc, go, ijpeg, li

表 2.4:　単体/ハイブリッド予測器の予測精度向上手法

手法	提案者	年	概要	特徴	実験環境	評価結果
Filtering 機構 [13]	Chang (Michigan 大)	1996	BTB の各エントリに Direction Bit と 2bit 飽和カウンタを追加 カウンタが飽和している時だけ，Direction bit の値を予測値として利用 その他は既存の予測器で予測	偏向の強い分岐命令を既存の予測器のエントリからはずすことで競合を緩和	トレース駆動型 SPECint95	2〜16KB において，通常の gshare 予測器より平均0.02%予測ミス率が低い
BTB 参照 [14]	斎藤 (早大)	2004	BTB にエントリを持たない分岐を常に NotTaken と予測	予測表の競合を緩和	SimpleScalar2.0/PISA SPECint95(train 入力)	8KB の gshare 予測器で平均 1.5%，1.5KB の Bi-Mode 予測器で平均 0.4%予測ミス率低減
SBI[17]	Manne (Compaq)	1999	2bit 飽和カウンタで予測信頼度を判定　信頼度が低い場合に予測を反転		SimpleScalar SPECint95	Combining 予測器，Bi-Mode 予測器に適用して，最大 0.5%予測ミス率低減
BPRU [18]-[20]	Juan (Catalunya 大)	2001	信頼度判定器は 2bit 飽和カウンタの表で，分岐予測，グローバル履歴，命令アドレス，予測したオペランド値でインデクス カウンタが閾値以上で予測反転	ソースオペランド値予測を利用して予測の信頼度を評価	SimpleScalar3.0c/Alpha SPECint2000	Alpha21264 の Combining 予測器に適用して，128KB で平均 1%予測ミス率低減

第3章　予測カウンタ状態の解析

第 2 章で紹介した分岐予測器の大半は，分岐条件の成立 (Taken)・不成立 (NotTaken) で遷移する予測カウンタ (2bit 飽和カウンタ) を利用して予測を行う．予測カウンタは，Strongly Taken, Weakly Taken, Weakly NotTaken, Strongly NotTaken の 4 状態を遷移する (図 3.1)．予測カウンタが Strongly Taken と WeaklyTaken 状態の場合には”Taken”と予測し，Strongly NotTaken と Weakly NotTaken 状態の場合には”NotTaken”と予測する．

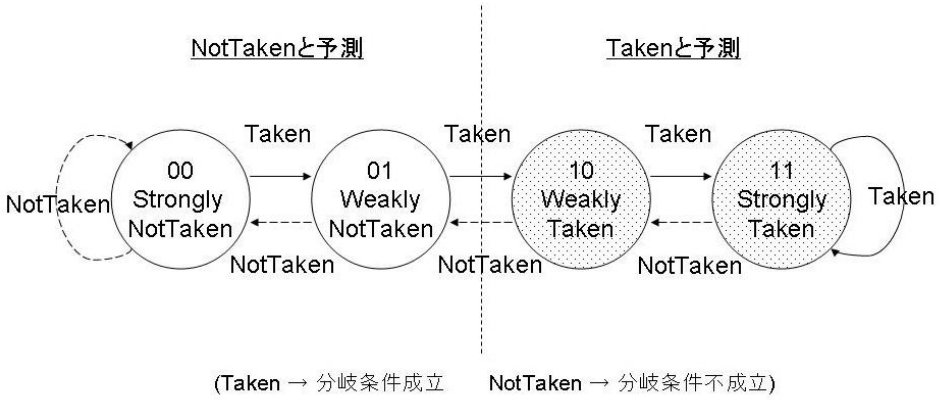


図 3.1: 2bit 飽和カウンタの状態遷移図

Strongly Taken 状態は，Strongly Taken 状態あるいは Weakly Taken 状態から遷移する．

- Strongly Taken 状態 → ”Taken”と予測 → 予測成功(結果は”Taken”) → Strongly Taken 状態
- Weakly Taken 状態 → ”Taken”と予測 → 予測成功(結果は”Taken”) → Strongly Taken 状態

つまり，予測カウンタが Strongly Taken 状態であるということは，前回の予測が成功していることを示す．Strongly NotTaken 状態も同様のことが言える．

逆に，Weakly Taken 状態は，Strongly Taken 状態あるいは Wekaly NotTaken 状態から遷移する．

- Strongly Taken 状態 → ”Taken” と予測 → 予測失敗(結果は”NotTaken”) → Weakly Taken 状態
- Weakly NotTaken 状態 → ”NotTaken” と予測 → 予測失敗(結果は”Taken”) → Wekaly Taken 状態

つまり，予測カウンタが Weakly Taken 状態であるということは，前回の予測が失敗していることを示す．Weakly NotTaken 状態も同様のことが言える．

以上から，予測カウンタの 4 状態は予測方向を決定するだけでなく，予測の信頼度を示していると推察できる．そこで，本節では予測カウンタの各状態 (Strongly/Weakly 状態) における予測精度を調査する．

3.1 実験環境

シミュレータは，SimpleScalar 3.0c/PISA[23] の sim-bpred を利用した．sim-bpred シミュレータでは，命令は逐次実行され，前の分岐が解決するまでは次の分岐は実行されない．ベンチマークは SPECint95(train 入力) を採用した．プログラムは gcc 2.7.2.3 -O2 -funroll-loops でコンパイルされている．

本実験では， BTB(Branch Target Buffer) のエントリ有無を参照した分岐予測手法 [14](第 2.3 節参照) を全ての分岐予測器に適用した．この手法の適用により，BTB にエントリのない分岐は分岐条件不成立 (NotTaken) と予測され，PHT における競合を削減できる．

分岐予測器は，代表的な単体予測器である bimodal 予測器 [1]，SAg 予測器 (2 レベル適応型予測器)[2]，gshare 予測器 [3] を対象にした．容量は，1KB，8KB，16KB の 3 種類を基準値に設定した．SAg 予測器に関しては，予測器を構成するテーブルのパラメータの関係で，1.2KB，7.5KB，15.5KB となる．各予測器のパラメータを表に示す．

表 3.1: 各ベンチマークの動的命令数 (100 万命令)，動的条件分岐命令数 (100 万命令)

プログラム	入力	全命令数	全条件分岐命令数
099.go	50 9 2stone9.in	554	63
124.m88ksim	Ctl.in	114	14
126.gcc	Amptjp.i	1,280	194
129.compress	1000 q 2131	36	4
130.li	train.lsp	182	24
132.ijpeg	vigo.ppm	1,407	87
134.perl	jumble.pl	2,293	299
147.vortex	persons.250	2,608	287

表 3.2: 各予測器のパラメータ (エントリ数)

予測器	容量 (KB)	パラメータ
bimodal	1.0	PHT 1K
	8.0	PHT 32K
	16.0	PHT 64K
gshare	1.0	PHT 1K
	8.0	PHT 32K
	16.0	PHT 64K
SAg	1.2	PHT 2K, BHT 0.5K*11bit
	7.5	PHT 16K, BHT 2K*14bit
	15.5	PHT 32K, BHT 4K*15bit

3.2 各予測カウンタ状態の予測精度

各予測器の，Strongly 状態における予測成功数と予測ミス数，Weakly 状態における予測成功数と予測ミス数の割合を図 3.2-3.4 に占めす (実データは付録 A を参照)．また，各状態の頻度，予測ミス率，全予測ミスに占める割合と，全体 (Strongly 状態と Weakly 状態合わせて) の予測ミス率の全ベンチマークの平均値を表 3.3 に示す．

どの容量でも同じ傾向を示したので，ここでは 8KB における結果を考察する．まず，各予測器における結果をみてみると，全予測の 85%以上が Strongly 状態であることがわかる．これは文献 [36] にあるように，多くの分岐命令は Taken あるいは NotTaken のどちらかに偏っている (偏向がある) ためである．また，Strongly 状態に比べて，Weakly 状態における予測ミス率が非常に大きい．このことから，予測カウンタが Weakly 状態の時は，予測ミスしやすいこと，つまり，予測の信頼度が低いことがわかる．また，Weakly 状態においては，予測ミス率が高いために，頻度が約 10%と低いにも関わらず全体の予測ミスの約 25〜40%を占める．

次に 3 種類の予測器間の比較を行う．全体の予測ミス率は gshare 予測器，SAg 予測器，bimodal 予測器の順で低いのに対し，Weakly 状態における予測ミス率と全体のミスに占める割合はその順で高い．特に，gshare 予測器と SAg 予測器では，Strongly 状態と Weakly 状態の予測ミス率の差が約 30%と大きい．このことから，gshare 予測器と SAg 予測器では，予測カウンタが Strongly 状態の場合には予測が成功し，Weakly 状態の場合には予測が失敗する傾向が特に強いと言える．gshare 予測器と SAg 予測器では，gshare 予測器の方がよりその傾向が強く，全体の予測ミス率も低い．

以上から，予測カウンタが Weakly 状態の場合には予測ミスしやすい，つまり，予測の信頼度が低いことがわかった．また，gshare，bimodal，SAg 予測器の中では，gshare 予測器の Weakly 状態が一番予測信頼度が低く，gshare 予測器の予測カウンタ状態がより正確な予測信頼度を示していると言える．

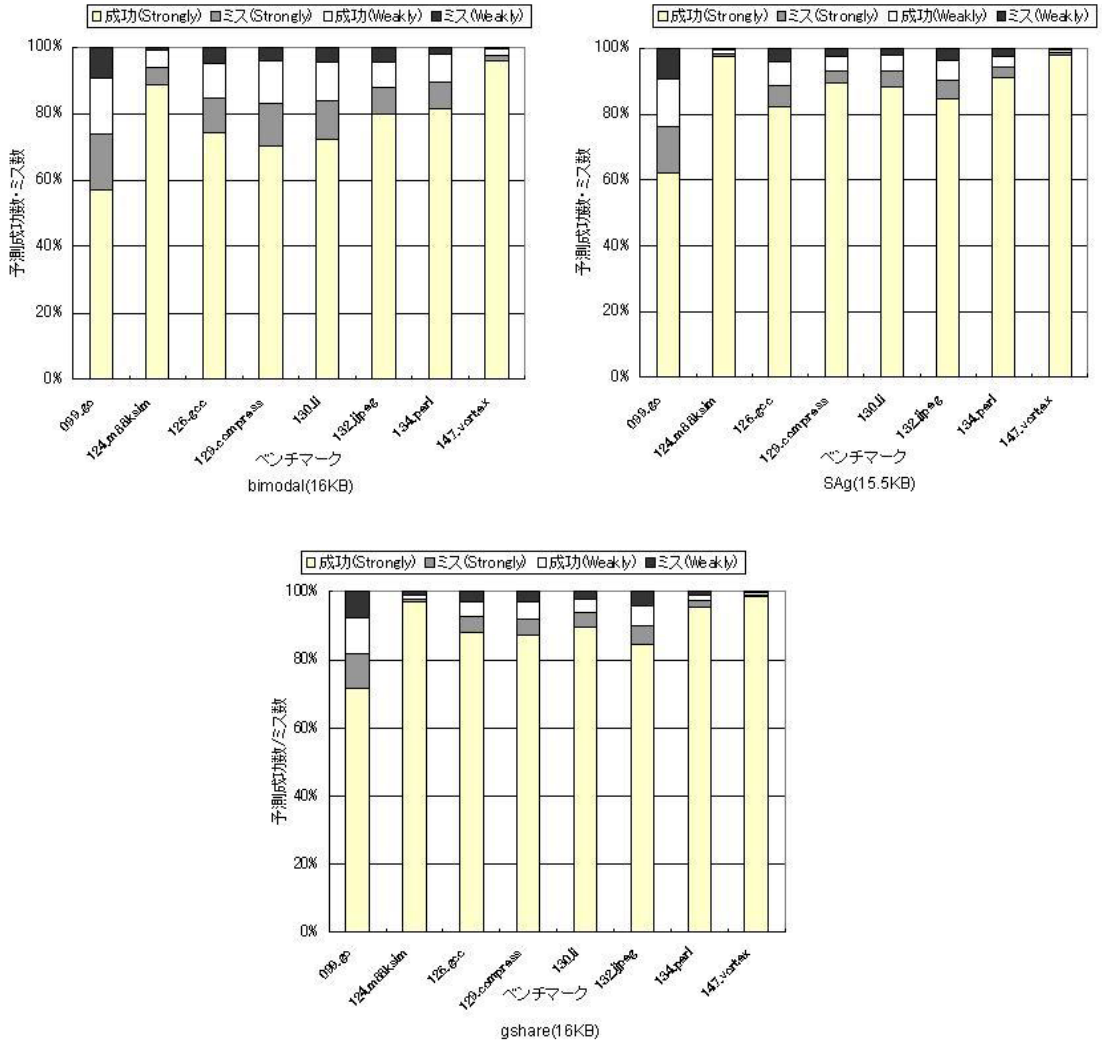

□成功(Strongly) ■ミス(Strongly) □成功(Weakly) ■ミス(Weakly)
予測成功数・ミス数
100%
80%
60%
40%
20%
0%
099.go
124.m88ksim
126.gcc
129.compress
130.li
132.ijpeg
134.perl
147.vortex
ベンチマーク
bimodal(16KB)
SAg(15.5KB)
gshare(16KB)

図 3.2: 16KB

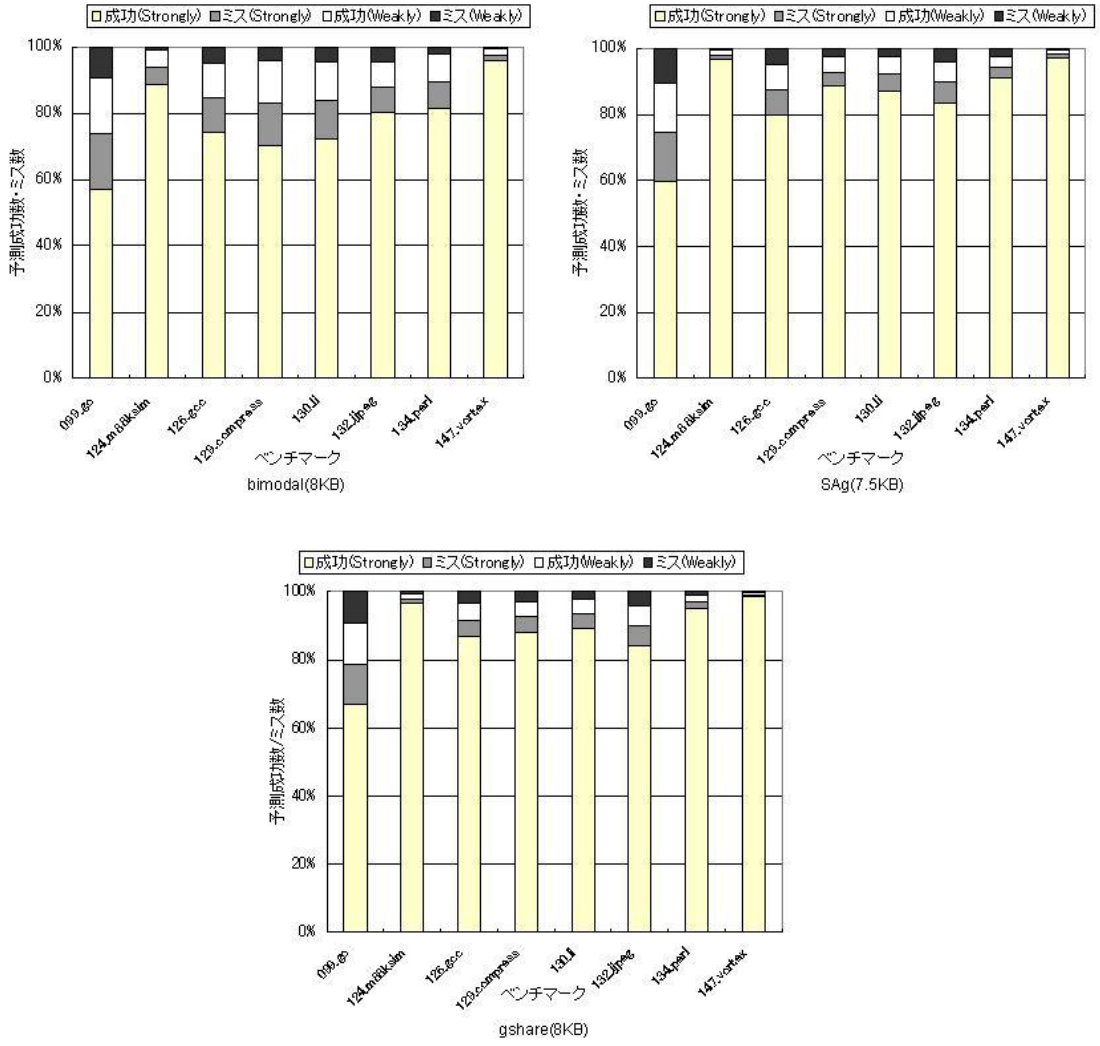

□成功(Strongly) ■ミス(Strongly) □成功(Weakly) ■ミス(Weakly)
予測成功数・ミス数
100%
80%
60%
40%
20%
0%
099.go
124.m88ksim
126.gcc
129.compress
130.li
132.ijpeg
134.perl
147.vortex
ベンチマーク
bimodal(8KB)
SAg(7.5KB)
gshare(8KB)

図 3.3: 8KB

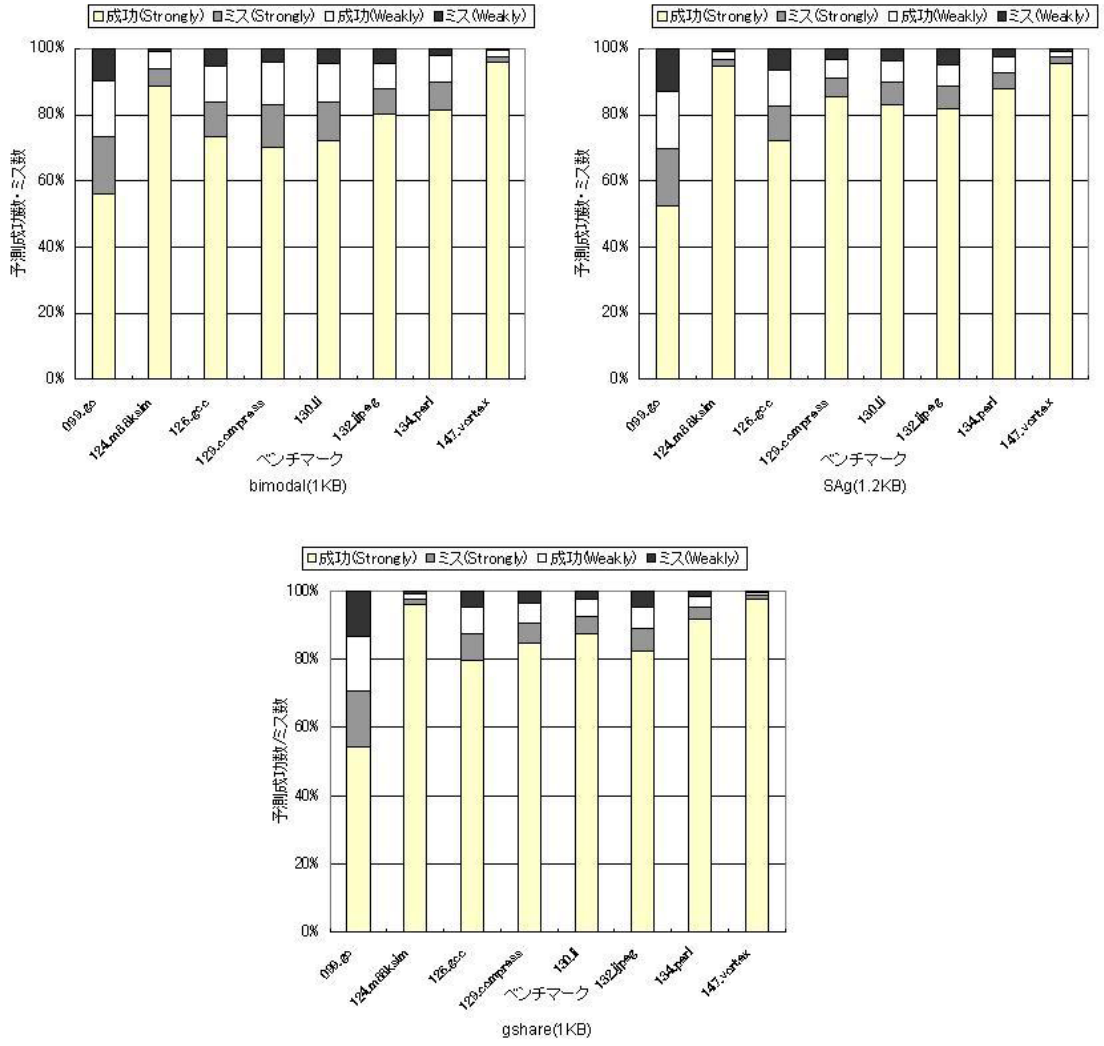

□成功(Strongly) ■ミス(Strongly) □成功(Weakly) ■ミス(Weakly)
予測成功数・ミス数
100%
80%
60%
40%
20%
0%
099.go
124.m88ksim
126.gcc
129.compress
130.li
132.ijpeg
134.perl
147.vortex
ベンチマーク
bimodal(1KB)
SAg(1.2KB)
gshare(1KB)

図 3.4: 1KB

表 3.3: 各予測カウンタ状態における使用率 (%)，予測ミス率 (%)，全体のミスに対する割合 (%)(平均)

容量	予測器	状態	使用率	予測ミス率	ミス割合	全体のミス率
16KB	bimodal	S	86.79	11.24	73.40	13.20
		W	13.21	26.60	26.60	
	SAg	S	91.64	5.88	62.40	8.23
		W	8.36	36.76	37.60	
	gshare	S	93.14	4.55	61.20	6.80
		W	6.86	38.39	38.80	
8KB	bimodal	S	86.80	11.24	73.40	13.20
		W	13.20	26.60	26.60	
	SAg	S	90.96	6.47	62.66	8.96
		W	9.04	36.87	37.34	
	gshare	S	92.45	5.11	61.72	7.50
		W	7.55	37.98	38.28	
1KB	bimodal	S	86.64	11.34	73.12	13.36
		W	13.36	26.89	26.88	
	SAg	S	88.59	8.49	63.12	11.40
		W	11.41	36.84	36.88	
	gshare	S	90.09	7.17	62.89	9.89
		W	9.91	37.04	37.11	

S:Strongly, W:Weakly

第4章　予測カウンタ状態に着目した予測選択手法Confidece-Selector

現在まで，様々な分岐予測器が提案されてきた．その中でも，複数の予測器を組み合わせたハイブリッド予測器 [3, 6, 7, 8, 9, 10, 11] は，高精度な予測器であることが知られている．ハイブリッド予測器には，各予測器による予測の中から，最終的な予測を選択する仕組みが必要となる．主なハイブリッド予測器は，予測器とは別に Selector を持っている．その Selector を用いて，各予測器の予測の中から最終的な予測を選択する．予測選択手法には，各予測器の予測成功・失敗の頻度 [3] や，分岐の偏向に基づく手法 [6] がある (第 2.2 節参照)．

しかし，Selector の精度は必ずしも高いわけではない．例えば，代表的なハイブリッド予測器である Combining 予測器 [3] では，SPECint95 ベンチマークで，全予測ミスのうち，選択ミス[1]が 2〜6 割を占める [37]．

分岐予測器は有限のハードウェアで如何に予測精度を向上させるかが重要である．ハイブリッド予測器は，複数の予測器と Selector で構成されるため，個々の構成要素に割り当てられる容量は単一の単体予測器と比べて小さくなる．そのため，わざわざ Selector を追加して予測選択を行うのは最善の手法とは言えない．

本研究では，予測器の予測カウンタ状態を新たに予測選択の指標に取り入れ，Selector の除去を試みる．具体的には，2 つの異なる予測器で構成されるハイブリッド予測器を前提に，一方の予測器の予測カウンタ状態を参照し，予測カウンタが強偏向 (Strongly) 状態の場合にはその予測器の予測を採用し，弱偏向 (Weakly) 状態の場合にはもう一方の予測器の予測を採用する．1 つの予測器に，予測選択権を与えることで従来の Selector を除去することでき，Selector に要していたハードウェアを各予測器に再配分することができる．

本章では，提案手法，実験環境，実験結果と考察を示す．

4.1　Confidence-Selector

第 3.2 節で示したように，bimodal 予測器 [1]，SAg 予測器 [2]，gshare 予測器 [3] の単体予測器は，予測カウンタの状態によって予測信頼度が異なる．本研究では，予測カウンタ

[1]正しく選択していればミスをしなかった予測

の信頼度に基づいた予測選択手法を提案する．以降，この手法を Confidence-Selector と呼ぶ．

Confidence-Selector 手法を Combining 予測器 [3] に適用する．Combining 予測器の構成要素には，グローバル履歴を利用する gshare 予測器 [3] と，PC 毎のローカル履歴を利用する SAg(2 レベル適応型) 予測器 [2] を採用する．SAg 予測器は，ローカル履歴を利用する 2 レベル適応型予測器の中でも，比較的小容量で，かつ，高精度な分岐予測器である [3]．

gshare 予測器は，Weakly 状態の時の予測精度が一番低く，かつ全体の予測ミス率も一番高い (第 3.2 節参照)．そのため，本方式では gshare 予測器の予測カウンタ状態を予測選択の指標に利用する．Confidence-Selector 手法を適用した Combining 予測器 (以下，Combining-CS 予測器と呼ぶ) を図 4.1 に示す．gshare 予測器の予測カウンタが Strongly(Strongly Taken/Strongly Not-taken) 状態の場合には，gshare 予測器の予測をそのまま利用する．つまり，Strongly Taken 状態の時は「Taken(分岐する)」，Strongly Not-taken 状態の時は「NotTaken(分岐しない)」と予測する．gshare 予測器の予測カウンタが Weakly (Weakly Taken / Weakly Not-taken) 状態の場合には，SAg 予測器の予測を採用する．SAg 予測器の BHT は常時更新するが，PHT は予測が採用された時だけ更新する．Combining-CS 予測器では，gshare 予測器が，実際の予測を行う予測器と，2 つの予測の中から最終的な予測を選択する Selector の両方を兼ねる．そのため，従来の Combining 予測器における Selector を省け，Selector に割り当てていたハードウェア量を gshare 予測器や SAg 予測器に割り当てられる．

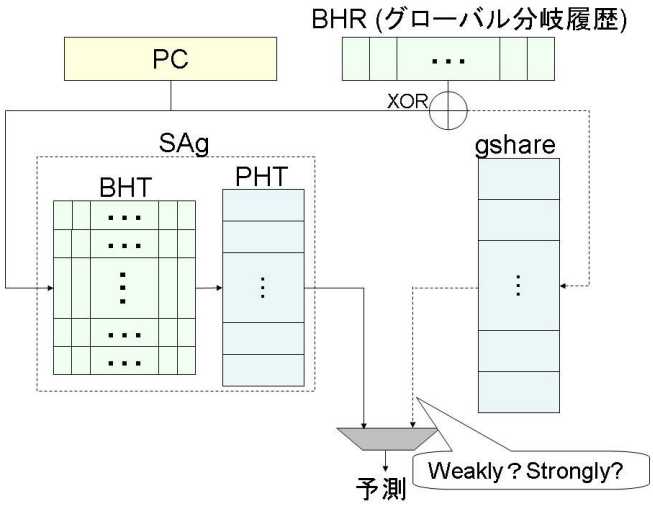


図 4.1: Confidence-Selector 手法を適用した Combining 予測器 (Combining-CS 予測器)

4.2 実験環境

シミュレータは，SimpleScalar 3.0c/PISA[23] の sim-bpred を利用した．sim-bpred シミュレータでは，命令は逐次実行され，前の分岐が解決するまでは次の分岐は実行されない．ベンチマークは SPECint95(train 入力) を採用した．プログラムは gcc 2.7.2.3 -O2 -funroll-loops でコンパイルされている．

本実験では， BTB(Branch Target Buffer) のエントリ有無を参照した分岐予測手法 [14](第 2.3.1 節参照) を全ての分岐予測器に適用した．この手法の適用により，BTB にエントリのない分岐は分岐条件不成立 (NotTaken) と予測され，予測表における競合を削減できる．

ハードウェア容量 1.5KB，12KB，24KB を基準に評価を行った．比較対象の予測器は，gshare 予測器 [3]，Bi-Mode 予測器 [6]，SAg 予測器 [2] と gshare 予測器を組み合わせた Combining 予測器 [3] である．各予測器のエントリ数等のパラメータを表 4.1 に示す．

Combining 予測器では，テーブル数の多い SAg 予測器を利用するため，Selector の容量を小さめにした．Combining 予測器では，複数のパラメータの組み合わせの中から，基準のハードウェア量以下で，かつベンチマーク全体で一番精度の良いパラメータを選択した (最適なパラメータに関しては付録 B を参照)．12KB と 24KB の基準値では，Combining 予測器は 11.75KB と 23KB となる．次節以降では，これらの容量を便宜的に 12KB，24KB と表現する．

表 4.1: 各予測器のパラメータ (PHT,BHT のエントリ数)

予測器	実容量 (KB)	パラメータ
Gshare	1.0	PHT 1K
	8.0	PHT 32K
	16.0	PHT 64K
BiMode	1.5	Choice/Taken/NotTakenPHT 各 2K
	12	Choice/Taken/NotTakenPHT 各 16K
	24	Choice/Taken/NotTakenPHT 各 32K
Combining	1.5	SAg: PHT 1K, BHT 0.5K*8bit gsahre: 2K, Selector: 1K
	11.75	SAg: PHT16K, BHT 1K*14bit gshare: 16K, Selector: 8K
	23	SAg: PHT16K, BHT 4K*14bit gshare: 32K, Selector: 16K

4.3 実験結果

まず，Confidence-Selector 手法を適用した Combining 予測器 (Combining-CS 予測器) の最適な構成 (パラメータ) を決定する．その後，Combining-CS 予測器と従来のハイブリッド予測器の予測精度の比較を行う．また，Selector の予測精度への影響を考察する．

4.3.1 最適なパラメータ

Combining-CS 予測器は，Selector を除いた Combining 予測器と同じ構成であり，構成要素には複数のパラメータが考えられる．本節では，Combining-CS 予測器における最適な構成パラメータを決定する．各容量 (1.5KB，12KB，24KB) 以下で，実験に用いたパラメータを表 4.2 に示す．各パラメータにおける全ベンチマーク平均の予測ミス率を表 4.3 に示す (各ベンチマークの予測ミス率は付録 C を参照)．

Combining-CS 予測器においては，gshare 予測器と SAg 予測器のどちらに多くハードウェアを割り当てるかが問題となる．実験の結果，gshare 予測器に多くハードウェアを割り当てた方が予測ミス率が低いことがわかった．Combining-CS 予測器では，gshare 予測器の予測カウンタ状態を予測選択の指標としているので，gshare 予測器の精度が予測選択精度を大きく左右する．gshare 予測器にハードウェアを多く割り当てた方が全体の予測ミス率も低いことが確認できた．次節の予測精度の比較では，一番予測ミス率の低かったパラメータで比較を行う．

表 4.2: 各予測器のパラメータ (PHT,BHT のエントリ数)

基準値 (KB)	番号	実容量 (KB)	gshare	SAg	
				BHT	PHT
1.5	1.5-A	1.5	2K	1K * 6bit	1K
	1.5-B	1.38	2K	0.5K * 10bit	1K
	1.5-C	1.5	2K	2K * 3bit	1K
	1.5-D	1.5	2K	0.5K * 8bit	2K
	1.5-E	1.5	2K	1K * 4bit	2K
	1.5-F	1.31	2K	0.25K * 10bit	2K
	1.5-G	1.5	4K	0.5K * 4bit	1K
12	12-A	11.5	16K	2K * 14bit	16K
	12-B	12.0	16K	4K * 8bit	16K
	12-C	12.0	16K	4K * 12bit	8K
	12-D	11.68	32K	1K * 13bit	8K
	12-E	12.0	32K	2K * 12bit	4K
24	24-A	23.5	32K	4K * 15bit	32K
	24-B	24.0	32K	8K * 12bit	16K
	24-C	23.5	64K	2K * 14bit	16K
	24-D	24.0	64K	4K * 8bit	16K
	24-E	24.0	64K	4K * 12bit	8K

表 4.3: 各予測器の平均予測ミス率%

基準値 (KB)	番号	予測ミス率 (%)
1.5	1.5-A	8.22
	1.5-B	8.33
	1.5-C	8.20
	1.5-D	8.24
	1.5-E	8.16
	1.5-F	8.39
	*1.5-G	7.78
12	12-A	6.24
	12-B	6.31
	12-C	6.33
	*12-D	5.88
	*12-E	5.94
24	24-A	5.65
	24-B	5.76
	*24-C	5.29
	*24-D	5.37
	*24-	5.37

*gshare 予測器の容量を大きくしたもの

4.3.2 予測精度

各容量，各ベンチマークにおける Bi-Mode 予測器 [6]，Combining 予測器 [3]，Combining-CS(Confidece-Selector 手法を適用した Combining) 予測器の予測ミス率 (%) を図 4.2 に示す (実データは付録 D 参照)．また，Combining-CS 予測器と，Combining 予測器，Bi-Mode 予測器の予測ミス率 (%) の差分を表 4.4 にまとめる．差分が負の値であれば，Combining 予測器，Bi-Mode 予測器に対して予測ミス率が低減したことを示す．逆に正の値であれば，予測ミス率が増加したことを示している．Combining-CS 予測器は，Combining 予測器と比較して，1.5KB では平均 0.06%，12KB では平均 0.22%，24KB では平均 0.31%予測ミス率が低減した．Bi-Mode 予測器と比較して，1.5KB では平均 0.02%，12KB では平均 0.41%，24KB では平均 0.50%予測ミス率が低減した．

容量が大きいほど，予測ミス率を低減できている．小容量の場合には，個々の構成要素 (Selector と各予測器) に割り当てられる容量そのものが小さい．そのため，Selector 部分のハードウェアを他の予測器に再配分しても，予測器自身の予測精度がそれほど向上しなかったためと考えられる．

124.m88ksim と 129.compress の予測精度低下 (対 Combining 予測器)

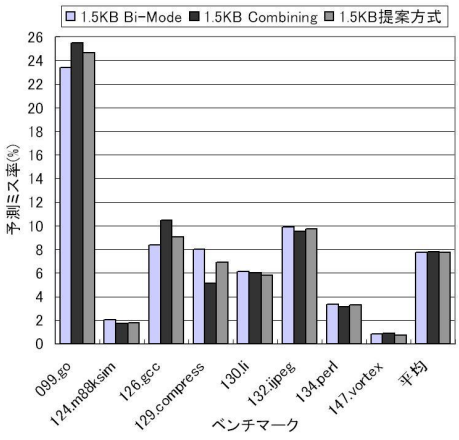
129.compress と 124.m88ksim は，全容量で Combining 予測器よりも予測ミス率が増加した．その原因を，第 3.2 節で実験した SAg 予測器と gshare 予測器における全体の予測ミス率 (表 4.5) から検討してみる．1KB の 124.m88ksim を除く全ての容量において SAg 予測器の方が予測ミス率が低い．第 3.2 節では，SAg と gshare 予測器はどちらも，Strongly 状態と Weakly 状態における予測ミス率に大きな差があることを示した．しかし，gshare 予測器の方が，全ベンチマーク平均の全体の予測ミス率が低いということで，Combining-CS 予測器では gshare 予測器の予測カウンタ状態を予測選択の指標に利用した．しかし，124.m88ksim と 129.compress は，gshare 予測器よりも SAg 予測器の方が予測精度が高い．そのため，これらのベンチマークに対しては，gshare 予測器ではなく，SAg 予測器の予測カウンタ状態を予測選択の指標に取り入れた方が精度を向上できたと考えられる．

第 4.3.3 節で SAg 予測器の予測カウンタ状態に基づいた予測選択手法について検討する．

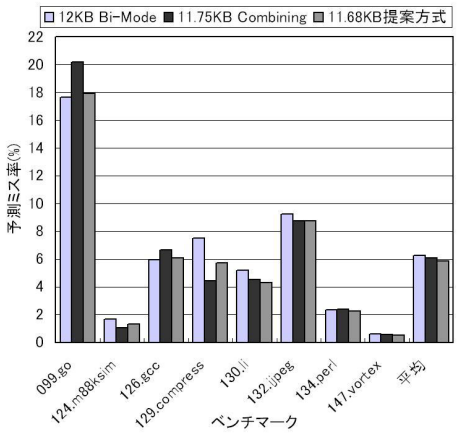
099.go と 126.gcc の予測精度低下 (対 Bi-Mode 予測器)

099.go と 126.gcc の予測ミス率は，Combining 予測器と比較すると低減したが，Bi-Mode 予測器と比較すると増加した (24KB の 099.go を除く)．099.go と 126.gcc は gshare 予測器におけるエントリ競合が激しいため，gshare 予測器の競合緩和を目的とした Bi-Mode 予測器が有効であることが知られている [6]．しかし，容量が大きくなれば競合は緩和され

るので，Combining-CS 予測器でも，容量が大きい場合には Bi-Mode 予測器との予測ミス率の差は小さい．24KB においては，099.go では Combining-CS 予測器の方が予測ミス率が低い．126.gcc についても+0.01%であり，差はほとんどない．



a) 1.5KB



b) 12KB

c) 24KB

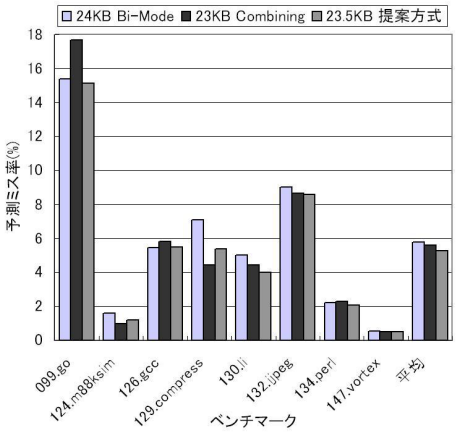


図 4.2: 1.5KB, 12KB, 24KB における Bi-Mode 予測器, Combining 予測器, Combining-CS 予測器の予測ミス率 (%)

表 4.4: Combining 予測器，Bi-Mode 予測器に対する予測ミス率の差分 (%)

容量	比較予測器	go	m88ksim	gcc	compress	li	ijpeg	perl	vortex	平均
1.5KB	Combining	-0.83	*0.05*	-1.4	*1.76*	-0.21	*0.2*	*0.17*	-0.19	-0.06
	Bi-Mode	*1.26*	-0.3	*0.66*	-1.1	-0.32	-0.19	-0.06	-0.09	-0.02
12KB	Combining	-2.28	*0.25*	-0.61	*1.31*	-0.25	-0.01	-0.15	-0.04	-0.22
	Bi-Mode	*0.28*	-0.35	*0.1*	-1.78	-0.88	-0.48	-0.09	-0.06	-0.41
24KB	Combining	-2.53	*0.22*	-0.34	*0.95*	-0.46	-0.06	-0.25	0.0	-0.31
	Bi-Mode	-0.21	-0.43	*0.01*	-1.72	-1.02	-0.43	-0.16	-0.06	-0.50

表 4.5: SAg 予測器と gshare 予測器の予測ミス率 (%)(124.m88ksim と 129.compress)

124.m88ksim				129.compress			
予測器	1KB	8KB	16KB	予測器	1KB	8KB	16KB
SAg	3.21	1.85	1.50	SAg	8.74	6.83	6.36
gshare	2.37	1.94	1.92	gshare	9.48	8.43	7.83

4.3.3 予測選択指標の違い

第 4.3.2 節で，124.m88ksim と 129.compress の予測精度が低下した理由について考察した．この 2 つのプログラムでは，gshare 予測器よりも SAg 予測器の方が予測精度がよい．そこで，SAg 予測器の予測カウンタ状態を予測選択の指標とした方式について実験を行った．SAg 予測器の予測カウンタが Strongly 状態の場合には SAg 予測器の予測を，Weakly 状態の場合には gshare 予測器の予測を採用する．SAg 予測器は常時更新するが，gshare 予測器の予測表は予測が採用された時だけ更新する．しかし，BHR(Branch History Registor) は常時更新する．この方式を Combining-CS-sag 予測器と呼ぶ．

Combining-CS-sag 予測器の予測精度は，SAg 予測器の予測精度に大きく依存するので，SAg 予測器に割り当てる容量を大きく設定する．実験に用いたパラメータを表 4.6 に示す．Combining-CS-sag 予測器においては，全容量の 3 分の 2 を SAg 予測器に割り当てた場合 (Combining-CS-sag1) と，全容量の大半を SAg 予測器に割り当てた場合 (Combining-CS-sag2) の両方を実験する．第 4.3.2 節の実験で予測精度が向上した 130.li(第 4.3.2 節参照) も参考として実験を行った．Combining 予測器，Combining-CS 予測器，Combining-CS-sag 予測器の予測ミス率を表 4.7 に示す．

124.m88ksim と 129.compress では，gshare 予測器の予測カウンタ状態を参照する (Combining-CS 予測器) よりも，SAg 予測器の予測カウンタ状態を参照する (Combining-CS-sag 予測器) 方が，予測ミス率が低い．逆に，130.li では，Combining-CS 予測器の方が予測ミス率が低い．つまり，SAg 予測器の方が gshare 予測器よりも予測精度が良いプログラムでは，SAg 予測器の予測カウンタ状態を予測選択の指標とした方が精度が良いことがわかる．

しかし，Combining-CS-sag 予測器でも，従来の Combining 予測器より予測ミス率が低減することはなかった．その原因を，再度，第 3.2 節の実験結果から検討する．124.m88ksim, 129.compres, 130.li の実験結果を表 4.8 に示す (表 A.2 からの抜粋)．130.li では，gshare 予測器が，全体の予測ミス率が一番低く，かつ，Strongly 状態と Weakly 状態の予測ミス率の差が一番大きい。つまり、gshare 予測器の予測カウンタは，他の予測器より的確に予測信頼度を示していると言える．しかし、124.m88ksim と 129.compress では、全体の予測ミス率は SAg 予測器が一番低いが、Strongly 状態と Weakly 状態の予測ミス率の差は gshare 予測器の方が大きい。つまり，SAg 予測器の予測カウンタは，gshare 予測器ほど的確に予測信頼度を示してはいないと考えられる．そのため，124.m88ksim や 129.compress のように，gshare 予測器より SAg 予測器の方が予測精度が良いプログラムでも，SAg 予測器の予測カウンタ状態を予測選択の指標に用いるのは有効ではないと考察できる．

表 4.6: 各予測器のパラメータ (PHT,BHT のエントリ数)

予測器	実容量 (KB)	パラメータ
Combining	11.75	SAg: PHT16K, BHT 1K*14bit gshare: 16K, Selector: 8K
Comb-CS	11.68	SAg: PHT8K, BHT 1K*13bit gshare: 32K
Comb-CS-sag1	11.5	SAg: PHT16K, BHT 2K*14bit gshare: 16K
Comb-CS-sag2	12.0	SAg: PHT16K, BHT 4K*12bit gshare: 8K

Combining 予測器と Combining-CS 予測器は第 4.3.2 節の実験と同じ

表 4.7: Combining, Combining-CS, Combining-CS-sag1, 2 予測器の予測ミス率 (%)

予測器	124.m88ksim	129.compress	130.li
Combining	1.07	4.42	4.54
Comb-CS	1.32	5.73	4.29
Comb-CS-sag1	1.25	5.25	5.27
Comb-CS-sag2	1.13	5.63	5.21

表 4.8: 各予測カウンタ状態における頻度 (%)，予測ミス率 (%)，全体のミスに対する割合 (%)，全体のミス率 (%)(8KB)

ベンチマーク	予測器	状態	頻度	予測ミス率	ミス割合	全体のミス率
124.m88ksim	bimodal	S	93.94	5.51	85.26	6.07
		W	6.06	14.74	14.74	
	SAg	S	98.08	1.28	67.84	1.85
		W	1.92	31.06	32.16	
	gshare	S	98.01	1.28	64.71	1.94
		W	1.99	34.44	35.29	
129.compress	bimodal	S	83.04	15.45	75.61	16.96
		W	16.96	24.39	24.39	
	SAg	S	92.84	4.68	63.64	6.83
		W	7.16	34.68	36.36	
	gshare	S	91.46	5.53	60.02	8.43
		W	8.54	39.50	39.98	
130.li	bimodal	S	83.93	13.97	72.98	16.07
		W	16.07	27.02	27.02	
	SAg	S	92.30	5.48	66.25	7.64
		W	7.70	33.48	33.75	
	gshare	S	93.54	4.44	64.64	6.43
		W	6.46	35.22	35.36	

S:Strongly, W:Weakly

4.3.4 Selector の影響

本節では，Combining 予測器の Selector と，Confidence-Selector(提案手法) の精度を比較する．Combining 予測器と Combinig-CS 予測器で，SAg 予測器と gshare 予測器の各パラメータを同一に設定する．Combining 予測器は，さらに数 KB の Selector を追加して予測を行い，Combining-CS 予測器は，追加のハードウェアなしで gshare 予測器の予測カウンタ状態に基づいて予測を行う．

SAg 予測器と gshare 予測器のパラメータを A:11.75KB と B:23KB の Combining 予測器と同一 (表 4.1 参照) に設定した．Combining-CS 予測器で必要なハードウェア量は Selector 部分を除いた A:9.75KB と B:19KB となる．

Combining 予測器と Combining-CS 予測器の予測ミス率を表 4.9 に示す．Combining-CS 予測器の予測ミス率は，Combining 予測器と比較して，A(9.75KB) では平均 0.18%，B(19KB) では平均 0.14%増加した．しかし，これは比較しているハードウェア量が異なることに注意する必要がある．Combining 予測器とハードウェア量を揃えた場合には，A(11.68KB) で平均 0.2%，B(23.5KB) で 0.3%予測ミス率は低下する (第 4.3.2 節参照)．Confidence-Selector 手法を適用しても，ただ Selector を取り除いただけの構成では予測精度は低下する．しかし，Selector に割り当てていたハードウェアを SAg 予測器や gshare 予測器に再配分することで，Combining 予測器よりも予測精度を向上できる．

099.go，130.li，134.perl は，A,B どちらの場合でも平均 0.16%予測ミス率が低減した．これらのベンチマークでは，Confidence-Selector 手法を適用し，ただ Selector を取り除くだけで予測精度が向上する．つまり，予測精度を低下させずに必要なハードウェア量を削減できる．

表 4.9: SAg 予測器と gshare 予測器のパラメータを同じにした場合の予測ミス率 (%)

予測器	go	m88ksim	gcc	compress	li	ijpeg	perl	vortex	平均
A:11.75KB Comb	20.21	1.07	6.68	4.42	4.54	8.76	2.40	0.57	6.08
A:9.75KB Comb-CS	19.94	1.37	6.7	5.89	4.31	8.95	2.38	0.57	6.26
B:23KB Comb	17.68	0.96	5.81	4.41	4.45	8.64	2.30	0.49	5.59
B:19KB Comb-CS	17.35	1.3	5.93	5.62	4.18	8.75	2.26	0.48	5.73

*Comb = Combining

4.4 プロセッサの処理性能に与える影響

Combining 予測器に Confidence-Selector 手法を適用したところ，24KB で平均約 0.3%の予測ミス率を低減できた．0.3%の予測ミス率低減がプロセッサの処理性能に与える影響を考察する．SPECint95(train 入力) において，従来の Combining 予測器を採用した場合の IPC と，Confidence-Selector 手法を適用して平均約 0.3%予測ミス率を低減した場合の IPC を算出し，向上率を求める．IPC を求める計算式は次のようになる．

$$
\begin{aligned}
Cycle &= \frac{N}{n} + Penalty \times Miss \\
IPC &= \frac{N}{Cycle}
\end{aligned}
$$

$Cycle$:総サイクル数, N:総命令数, $Penalty$:分岐ミスペナルティ, $Miss$:分岐ミス数, n:同時実行可能数 (パイプライン本数)

命令フェッチ幅を 4 命令とし，パイプライン 10 段プロセッサ，20 段プロセッサ，40 段プロセッサにおける IPC 向上率を図 4.3 に示す．予測ミス率を平均 0.3%低減することで，10 段プロセッサでは約 1.2%の IPC 向上を達成できる．パイプライン段数が深化するにつれ，分岐予測ミスペナルティは大きくなるので，予測ミス率低減による IPC 向上率は大きくなる．将来パイプライン段数が深化し，40 段プロセッサとなった場合，平均 0.3%の予測ミス率低減で約 4.0%の IPC 向上を達成できる．

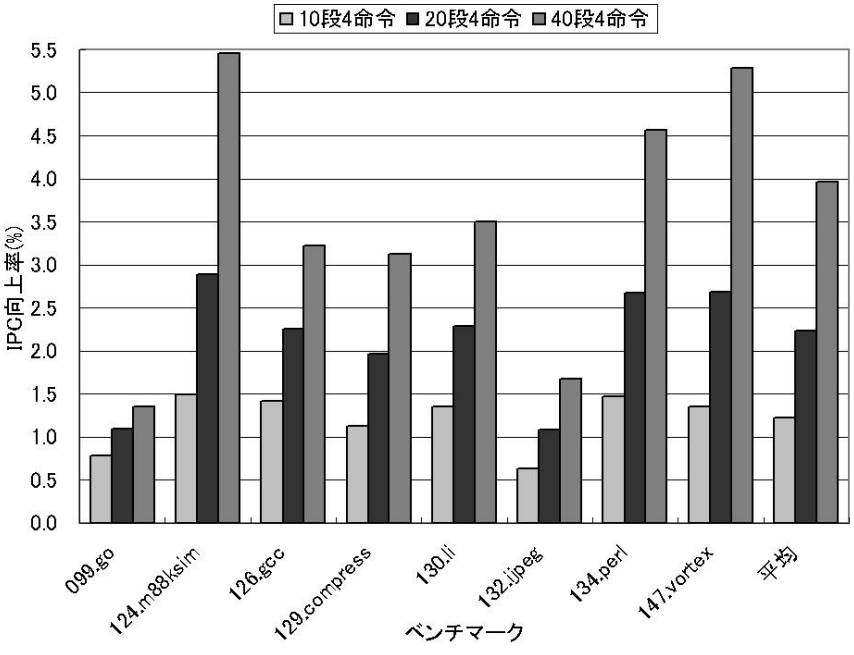


図 4.3: 平均 0.3%予測ミス率低減による IPC 向上率 (%)

4.5 実験のまとめ

本研究では，予測カウンタの予測信頼度に基づいた予測選択手法 Confidence-Selector を提案し，Combining 予測器 [3] に適用した (この予測器を Combining-CS 予測器と呼ぶ). 具体的には，gshare 予測器 [3] の予測カウンタ状態が Strongly 状態である場合には gshare 予測器の予測を採用し，Weakly 状態である場合には SAg 予測器 [2] の予測を採用する. gshare 予測器は，予測を行うだけでなく，従来のハイブリッド予測器の Selector の役割を持つ．Selector を排除することで，本来 Selector に割り当てる必要のあったハードウェアを，他の予測器に割り当てることができる.

SimpleScalar 3.0d/PISA の sim-bpred シミュレータを使い，SPECint95(train) を対象に実験を行った．その結果，11.68KB の Combining-CS 予測器は，11.75KB の Combining 予測器と比較して平均 0.22%予測ミス率が低減した．また，23.5KB の Combining-CS 予測器では，23KB の Combining 予測器と比較して平均 0.31%予測ミス率が低減した.

第5章　おわりに

近年のプロセッサは，命令の読み出し，解釈，データの読み出し，実行，実行結果の書き込み等を段階ごとに順次同時に行うパイプライン処理を行っている．しかし，プログラム中に分岐命令が存在すると，分岐先が確定するまで後続の命令を実行できず，パイプラインにストールが発生する．この制御依存によるパイプラインストールを緩和するために，多くのプロセッサでは分岐予測が採用されている．分岐予測により，分岐先以降の命令を投機的に実行でき，パイプラインストールを回避できる．しかし近年，パイプライン段数の増加に伴い，分岐予測ミスペナルティが増大している．そのため，分岐予測ミスの低減は，プロセッサの性能向上のために不可避な問題となっている．

現在まで，様々な分岐予測器が提案されてきた．その中でも，複数の予測器を組み合わせたハイブリッド分岐予測器は，高精度な予測器であることが知られている．ハイブリッド予測器には，各予測器による予測の中から最終的な予測を選択する仕組みが必要となる．多くのハイブリッド分岐予測器は，予測器とは別に Selector を用意し，その Selector により最終的な予測を決定する．しかし，Selector の精度はそれほど高くないという報告もある．また，Selector 用のハードウェアも必要となる．

本研究では，予測器の予測カウンタ状態毎に予測精度が異なることに着目し，予測カウンタの状態を参照した予測選択手法 Confidence-Selector を提案した．具体的には，2 つの予測器で構成されるハイブリッド予測器において，一方の予測器の予測カウンタ状態を参照し，予測カウンタが強偏向 (Strongly) 状態の場合にはその予測器の予測を採用する．弱偏向 (Weakly) 状態の場合にはもう一方の予測器の予測を採用する．本手法では，従来の Selector が不要となり，Selector に要していたハードウェアを各予測器に割り当てることが可能となる．

SimpleScalar 3.0d/PISA の sim-bpred シミュレータを使い，SPECint95(train) を対象に実験を行った．代表的なハイブリッド分岐予測器である Combining 予測器に適用したところ，12KB の容量で平均 0.22%，24KB で平均 0.31%予測ミス率が低減できた．パイプライン段数が深化した 4 命令同時フェッチ可能な 40 段プロセッサでは，平均 0.31%の予測ミス率低減により，約 4.0%の処理速度 (IPC) 向上を達成できる．

今後は，他のハイブリッド予測器にも Confidence-Selector 手法を適用できるか検討していきたい．

謝辞

本研究にあたり指導，助言を頂いた，指導教員の山名早人助教授，山名研究室斉藤史子氏に感謝の意を表する．

参考文献

[1] Smith J. E.: A Study of Branch Prediction Strategies, Proc. of 8th ISCA, pp. 135-148 (1981).

[2] Yeh T. Y. and Patt Y. N.: A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History, Proc. of 20th ISCA, pp. 257-266 (1993).

[3] McFarling S.: Combining branch predictors, Technical Report TN-36, Digital Western Research Laboratory (1993).

[4] Skadron K., Martonosi M. and Clark D. W.: A Taxonomy of Branch Mispredictions and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions, Proc. of 9th PACT, pp. 199-206 (2000).

[5] Sprangle E., Chappell R. S. et al: The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference, Proc. of 24th ISCA, pp. 284-291 (1997).

[6] Lee C. C., Chen I. K. and Mudge T. N.: The Bi-Mode Branch Predictor, Proc. of MICRO-30, pp. 4-13 (1997).

[7] Eden A. N. and Mudge T.N.: The YAGS Branch Prediction Scheme, Proc. of MICRO-31, pp. 69-77 (1998).

[8] Michaud P., Seznec A. and Uhlig R.: Trading Conflict and Capacity Aliasing in Conditional Branch Predictors, Proc. of 24th ISCA, pp. 292-303 (1997).

[9] Littlestone N. and Warmuth M. K.: The Weighted Majority Alogorithm, Information and Computation, Vol. 108, pp. 212-261 (1994).

[10] Loh G. H. and Henry D. S.: Predicting Conditional Branches with Fusion-Based Hybrid Predictors, Proc. of 11th PACT, pp. 165-176 (2002).

[11] Heil T.H., Smith Z. and Smith J. E.: Improving Branch Predictors by Correlating on Data Values, Proc. of MICRO-32, pp. 28-37 (1999).

[12] Lee J. K .F. and Smith A. J.: Branch Prediction Strategies and Branch Target Buffer Design, IEEE Computer, Vol. 17, No. 1, pp. 6-22 (1984).

[13] Chang P. Y., Evers M. and Patt Y. N.: Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference, Proc. of 5th PACT, pp. 48-57(1996).

[14] 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, 情報処理学会 ACS 論文誌, Vol.45, No. SGI 11(ACS 7), pp. 71-79 (2004).

[15] 吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣: Bimode-Plus 分岐予測器の提案, 信学技報 (CPSY2003), pp. 25-30 (2003).

[16] 吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣: 極端な偏りを利用する Bimode++分岐予測器, 情処研報 (2005-ARC-161), pp. 151-156 (2005).

[17] Manne S. Klauser A. and Grunwald D.: Branch Prediction using Selective Branch Inversion, Proc. of 8th PACT, pp. 48-56 (1999).

[18] Aragon J. L., Gonzalez J. et al.: Selective Branch Prediction Reversal by Correlating with Data Values and Control Flow, Proc. of ICCD'01, pp. 228-233 (2001).

[19] Gonzalez J., Gonzalez A.: Control-Flow Speculation through Value Prediction, IEEE Trans. on Computers, Vol. 50, No. 21, pp. 1362-1376 (2001)

[20] Aragon J. L., Gonzalez J., Garcia J. M. and Gonzalez A.: Confidence Estimation for Branch Prediction Reversal, Proc. of 8th HiPC, pp. 214-223 (2001).

[21] Haungs M., Sallee P. and Farrens M.: Branch Transition Rate: A New Metric for Improved Branch Classification Analysis, Proc. of 6th HPCA, pp. 241-50 (2000).

[22] M. Evers, S. J. Patel, R. S. Chappel, Y. N. Patt: An Analysis of Correlation and Predictability: What Makes Two-Level Branch Predictors Work, Proc. of 25th ISCA, pp. 52-61 (1998).

[23] Burger D. and Austin T. M.: The SimpleScalar Tool Set, Version2.0, Technical report (1997).

[24] Larson E., Chatterjee S. and Austin T.: MASE: A Novel Infrastructure for Detailed Microarchitecural Modeling, Proc. of ISPASS (2001).

[25] 斎藤史子, 山名早人: 命令レベル投機的実行に関する技術動向調査と分岐方向予測機構改良の提案, 早稲田大学卒業論文 (2002)

[26] 斎藤史子, 北村健志, 山名早人: ハイブリッド分岐方向予測機構の性能比較, 情処研報 (2003-ARC-150), pp. 89-94 (2002).

[27] Jimenez D. A. and Lin C.: Dynamic Branch Prediction with Perceptrons, Proceedings of the 7th HPCA, pp.197-206 (2001).

[28] Jimenez D. A. and Lin C.: Neural Methods for Dynamic Branch Prediction, ACM Transactions on Computer Systems, Vol. 20, No. 4, pp. 369-397 (2002).

[29] Uhlig R. et al.: Instruction Fetching: Coping with Code Bloat, Proc. of 22nd ISCA, pp. 345-356 (1995).

[30] Smith M. D.: Support for Speculative Execution in High-Performance Processors, PhD Thesis, Stanford University (1992).

[31] Woo S. C. et al.: The SPLASH-2 Programs: Characterization and Methodological Considerations, Proc. of 22th ISCA, pp. 24-36 (1995).

[32] Kessler R. E., McLellan E. J. and Webb D. A.: Alpha21264 マイクロプロセッサ・アーキテクチャ, 技術報告.

[33] Skadron K., Ahuja P. S., Martonosi M. and Clark D. W.: Branch Prediction, Instruction-Window Size, and Cache Size: Performance Trade-Offs and Simulation Techniques, IEEE Transactions on Computers, Vol. 48, No. 11, pp. 1260-1281 (1999).

[34] Juan T., Sanjeevan S. and Navarro J. J.: Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction, Proc. of 25th ISCA, pp. 155-166 (1998).

[35] M. Evers, S. J. Patel, R. S. Chappel and Y. N. Patt: An Analysis of Correlation and Predictability: What Makes Two-Level Branch Predictors Work, Proc. of 25th ISCA, pp. 52-61 (1998).

[36] M. Haungs, P. Sallee and M. Farrens: Branch Transition Rate: A New Metric for Improved Branch Classification Analysis, in Proc. of HPCA-6, pp. 241-250 (2000)

[37] 斎藤史子, 仲沢由香里, 山名早人: ハイブリッド予測機構における選択器と予測器の強調による予測ミス率の低減, 情処研報 (2003-ARC-154), pp. 115-120 (2003).

[38] Jimenez D. A. and Lin C.: Dynamic Branch Prediction with Perceptrons, Proceedings of the 7th HPCA, pp.197-206 (2001).

研究業績

[1] 斎藤史子, 仲沢由香里, 山名早人: ハイブリッド予測機構における選択器と予測器の強調による予測ミス率の低減, 情報処理学会研究会報告 計算機アーキテクチャ研究会 (2003-ARC-154), pp. 115-120 (2003).

[2] 仲沢由香里, 斎藤史子, 山名早人: 弱偏向状態に着目した分岐予測手法, 情報処理学会研究会報告 計算機アーキテクチャ研究会 (2005-ARC-161), pp. 145-150 (2005).

[3] 斎藤史子, 仲沢由香里, 山名早人: ハイブリッド分岐予測器を対象にした Confidence-Selector の提案, 情報処理学会研究会 先進的計算基盤システムシンポジウム (SACSIS), 情報処理学会研究会論文誌 コンピューティングシステム (ACS) 投稿中.

付 録A　予測カウンタの各状態における予測精度

予測カウンタの各状態における頻度 (%), 予測ミス率 (%), 全体のミスに占める割合 (%), 全体 (全状態合計) のミス率 (%) を表 A.1-A.3 に示す (第 3.2 節実験).

A.1　16KB

A.2　8KB

A.3　1KB

表 A.1: 各予測カウンタ状態における頻度 (%)，予測ミス率 (%)，全体のミスに対する割合 (%)，全体のミス率 (%)(16KB)

ベンチマーク	予測器	状態	頻度	予測ミス率	ミス割合	全体のミス率
099.go	bimodal	S	73.81	22.88	64.50	26.18
		W	26.19	35.50	35.50	
	SAg	S	76.33	18.46	59.91	23.53
		W	23.67	39.84	40.09	
	gshare	S	82.02	12.53	57.62	17.84
		W	17.98	42.06	42.38	
124.m88ksim	bimodal	S	93.94	5.51	85.26	6.07
		W	6.06	14.74	14.74	
	SAg	S	98.40	1.00	65.67	1.50
		W	1.60	32.30	34.33	
	gshare	S	98.02	1.14	58.34	1.92
		W	1.98	40.48	41.66	
126.gcc	bimodal	S	84.65	12.30	68.00	15.31
		W	15.35	31.92	32.00	
	SAg	S	88.85	7.64	61.26	11.08
		W	11.15	38.49	38.74	
	gshare	S	92.65	4.73	60.09	7.29
		W	7.35	39.56	39.91	
129.compress	bimodal	S	83.04	15.45	75.61	16.96
		W	16.96	24.39	24.39	
	SAg	S	93.12	4.14	60.57	6.36
		W	6.88	36.47	39.43	
	gshare	S	92.04	5.16	60.71	7.83
		W	7.96	38.66	39.29	
130.li	bimodal	S	83.93	13.97	72.98	16.07
		W	16.07	27.02	27.02	
	SAg	S	93.04	4.95	67.26	6.85
		W	6.96	32.23	32.74	
	gshare	S	93.76	4.25	64.27	6.21
		W	6.24	35.56	35.73	
132.ijpeg	bimodal	S	87.69	8.71	62.07	12.31
		W	12.31	37.94	37.93	
	SAg	S	90.40	6.55	61.84	9.58
		W	9.60	38.04	38.16	
	gshare	S	90.21	6.24	57.74	9.75
		W	9.79	42.09	42.26	
134.perl	bimodal	S	89.66	9.14	79.26	10.34
		W	10.34	20.74	20.74	
	SAg	S	94.39	3.43	57.79	5.61
		W	5.61	42.15	42.21	
	gshare	S	97.27	1.78	63.55	2.72
		W	2.73	36.31	36.45	
147.vortex	bimodal	S	97.63	1.94	79.52	2.38
		W	2.37	20.55	20.48	
	SAg	S	98.62	0.90	64.91	1.36
		W	1.38	34.58	35.09	
	gshare	S	99.14	0.58	67.27	0.85
		W	0.86	32.37	32.73	

S:Strongly, W:Weakly

表 A.2: 各予測カウンタ状態における頻度 (%)，予測ミス率 (%)，全体のミスに対する割合 (%)，全体のミス率 (%)(8KB)

ベンチマーク	予測器	状態	頻度	予測ミス率	ミス割合	全体のミス率
099.go	bimodal	S	73.80	22.89	64.48	26.19
		W	26.20	35.51	35.52	
	SAg	S	74.54	20.08	59.06	25.34
		W	25.46	40.75	40.94	
	gshare	S	78.80	15.26	56.97	21.11
		W	21.20	42.83	43.03	
124.m88ksim	bimodal	S	93.94	5.51	85.26	6.07
		W	6.06	14.74	14.74	
	SAg	S	98.08	1.28	67.84	1.85
		W	1.92	31.06	32.16	
	gshare	S	98.01	1.28	64.71	1.94
		W	1.99	34.44	35.29	
126.gcc	bimodal	S	84.63	12.31	67.95	15.33
		W	15.37	31.97	32.05	
	SAg	S	87.60	8.70	61.71	12.35
		W	12.40	38.11	38.29	
	gshare	S	91.78	5.35	60.12	8.17
		W	8.22	39.62	39.88	
129.compress	bimodal	S	83.04	15.45	75.61	16.96
		W	16.96	24.39	24.39	
	SAg	S	92.84	4.68	63.64	6.83
		W	7.16	34.68	36.36	
	gshare	S	91.46	5.53	60.02	8.43
		W	8.54	39.50	39.98	
130.li	bimodal	S	83.93	13.97	72.98	16.07
		W	16.07	27.02	27.02	
	SAg	S	92.30	5.48	66.25	7.64
		W	7.70	33.48	33.75	
	gshare	S	93.54	4.44	64.64	6.43
		W	6.46	35.22	35.36	
132.ijpeg	bimodal	S	87.74	8.68	62.10	12.26
		W	12.26	37.91	37.90	
	SAg	S	89.67	6.94	60.26	10.32
		W	10.33	39.68	39.74	
	gshare	S	89.99	6.40	57.68	9.99
		W	10.01	42.21	42.32	
134.perl	bimodal	S	89.66	9.14	79.26	10.34
		W	10.34	20.74	20.74	
	SAg	S	94.37	3.50	58.67	5.63
		W	5.63	41.30	41.33	
	gshare	S	96.98	2.01	64.65	3.01
		W	3.02	35.27	35.35	
147.vortex	bimodal	S	97.63	1.94	79.53	2.38
		W	2.37	20.54	20.47	
	SAg	S	98.26	1.12	63.84	1.73
		W	1.74	35.86	36.16	
	gshare	S	99.06	0.61	64.98	0.93
		W	0.94	34.76	35.02	

S:Strongly, W:Weakly

表 A.3: 各予測カウンタ状態における頻度 (%)，予測ミス率 (%)，全体のミスに対する割合 (%)，全体のミス率 (%)(1KB)

ベンチマーク	予測器	状態	頻度	予測ミス率	ミス割合	全体のミス率
099.go	bimodal	S	73.31	23.27	63.89	26.70
		W	26.69	36.12	36.11	
	SAg	S	69.77	24.75	57.18	30.19
		W	30.23	42.76	42.82	
	gshare	S	70.53	23.14	55.41	29.46
		W	29.47	44.58	44.59	
124.m88ksim	bimodal	S	93.85	5.61	85.45	6.16
		W	6.15	14.56	14.55	
	SAg	S	96.78	2.31	69.82	3.21
		W	3.22	30.06	30.18	
	gshare	S	97.61	1.71	70.56	2.37
		W	2.39	29.21	29.44	
126.gcc	bimodal	S	84.02	12.63	66.48	15.96
		W	15.98	33.47	33.52	
	SAg	S	82.72	12.88	61.68	17.27
		W	17.28	38.30	38.32	
	gshare	S	87.30	8.91	61.45	12.65
		W	12.70	38.42	38.55	
129.compress	bimodal	S	83.04	15.45	75.61	16.96
		W	16.96	24.39	24.39	
	SAg	S	91.21	6.19	64.58	8.74
		W	8.79	35.24	35.42	
	gshare	S	90.48	6.37	60.81	9.48
		W	9.52	39.00	39.19	
130.li	bimodal	S	83.93	13.97	72.98	16.07
		W	16.07	27.02	27.02	
	SAg	S	89.71	7.33	63.97	10.28
		W	10.29	36.00	36.03	
	gshare	S	92.34	5.45	65.79	7.65
		W	7.66	34.17	34.21	
132.ijpeg	bimodal	S	87.74	8.68	62.10	12.27
		W	12.26	37.92	37.90	
	SAg	S	88.54	7.56	58.42	11.46
		W	11.46	41.58	41.58	
	gshare	S	88.83	7.32	58.25	11.17
		W	11.17	41.73	41.75	
134.perl	bimodal	S	89.66	9.14	79.26	10.34
		W	10.34	20.74	20.74	
	SAg	S	92.72	5.05	64.33	7.28
		W	7.28	35.66	35.67	
	gshare	S	95.12	3.46	67.42	4.88
		W	4.88	32.57	32.58	
147.vortex	bimodal	S	97.60	1.96	79.16	2.41
		W	2.40	20.92	20.84	
	SAg	S	97.27	1.83	64.94	2.74
		W	2.73	35.14	35.06	
	gshare	S	98.51	0.96	63.39	1.49
		W	1.49	36.61	36.61	

S:Strongly, W:Weakly

付 録B　Combining予測器の最適なパラメータ

SAg 予測器と gshare 予測器を組み合わせた Combining 予測器の，1.5KB, 12KB, 24KB 容量における最適なパラメータを決定する (第 4.2 節実験)．パラメータの候補を表 B.1 に，各パラメータ時の予測ミス率を表 B.2 に示す．

最適なパラメータは，1.5KB-A，12KB-B，24KB-C である．

表 B.1: 各容量における Combining 予測器のパラメータ (PHT,BHT のエントリ数)

基準値 (KB)	番号	実容量 (KB)	Selector	gshare	SAg	
					BHT	PHT
1.5	1.5-A	1.5	1K	2K	0.5K * 8bit	1K
	1.5-B	1.5	1K	2K	0.5K * 4bit	2K
	1.5-C	1.5	1K	2K	1K * 4bit	1K
12	12-A	11.75	8K	16K	1K * 14bit	16K
	12-B	11.25	8K	16K	2K * 13bit	8K
	12-C	12.0	8K	16K	2K * 8bit	16K
	12-D	12.0	8K	16K	4K * 8bit	8K
24	24-A	23.0	16K	32K	4K * 14bit	16K
	24-B	23.25	16K	32K	2K * 13bit	32K

表 B.2: 各パラメータ時の Combining 予測器の予測ミス率 (%)

基準値	番号	go	m88ksim	gcc	compress	li	ijpeg	perl	vortex	平均
1.5KB	1.5-A	25.49	1.71	10.46	5.14	6.03	9.52	3.15	0.92	7.80
	1.5-B	25.01	1.69	9.90	6.01	6.23	9.59	3.16	0.92	7.81
	1.5-C	25.00	1.66	9.94	6.05	6.16	9.75	3.16	0.90	7.83
12KB	12-A	20.21	1.00	6.68	4.42	4.54	8.76	2.40	0.57	6.07
	12-B	19.84	1.07	6.56	4.66	4.84	9.05	2.39	0.56	6.12
	12-C	19.89	0.94	6.46	4.77	4.90	8.88	2.40	0.52	6.10
	12-D	19.88	0.99	6.43	4.84	4.89	8.93	2.41	0.52	6.11
24KB	24-A	17.68	0.96	5.81	4.41	4.45	8.64	2.30	0.49	5.59
	24-B	17.88	0.87	5.93	4.65	4.13	8.54	2.35	0.49	5.61

付 録C　提案方式(Combining-CS予測器)の最適なパラメータ

Confidence-Selector 手法を適用した Combining 予測器 (Combining-CS 予測器) の各パラメータにおける予測ミス率を示す (第 4.3.1 節実験).

表 C.1: 各パラメータ時の提案方式の予測ミス率 (%)

予測器	go	m88ksim	gcc	compress	li	ijpeg	perl	vortex	平均
1.5KB-A	26.02	1.84	10.49	7.00	5.97	10.06	3.48	0.78	8.21
1.5KB-B	26.47	1.90	10.84	6.83	6.08	10.16	3.50	0.79	8.32
1.5KB-C	25.64	1.80	10.33	7.04	6.27	10.06	3.47	0.78	8.17
1.5KB-D	26.38	1.86	10.63	6.82	5.87	10.01	3.50	0.79	8.23
1.5KB-E	25.55	1.89	10.28	7.03	6.14	10.01	3.51	0.80	8.15
1.5KB-F	27.09	1.88	11.03	6.77	5.86	10.13	3.48	0.81	8.38
1.5KB-G	24.66	1.76	9.06	6.90	5.82	9.92	3.32	0.77	7.78
12KB-A	19.70	1.50	6.62	5.89	4.31	8.94	2.45	0.55	6.25
12KB-B	19.49	1.60	6.49	6.27	4.72	9.13	2.45	0.57	6.34
12KB-C	19.85	1.48	6.59	6.10	4.45	9.23	2.40	0.58	6.34
12KB-D	17.93	1.32	6.07	5.73	4.29	8.95	2.25	0.53	5.88
12KB-E	17.95	1.32	6.03	5.85	4.46	9.10	2.26	0.53	5.94
24KB-A	17.09	1.29	5.90	5.55	4.09	8.51	2.25	0.52	5.65
24KB-B	17.26	1.24	5.90	5.82	4.22	8.88	2.22	0.51	5.76
24KB-C	15.15	1.18	5.47	5.36	3.99	8.58	2.05	0.50	5.29
24KB-D	15.01	1.18	5.38	5.76	4.39	8.79	1.98	0.49	5.37
24KB-E	15.27	1.18	5.45	5.58	4.11	8.87	2.03	0.50	5.37

*gshare 予測器の容量を大きくしたもの

付 録D　提案方式(Combining-CS予測器)と従来予測器の予測精度の比較

各容量における gshare 予測器, Bi-Mode 予測器, Combining 予測器と提案方式 (Combining-CS 予測器) の予測ミス率を表 D.1 に示す (第 4.3.2 節実験).

表 D.1: 各予測器の予測ミス率 (%)

予測器	go	m88ksim	gcc	compress	li	ijpeg	perl	vortex	平均
1.0KB gshare	27.22	2.03	10.28	8.06	6.08	10.5	3.74	3.94	8.98
1.5KB Bi-Mode	23.40	2.06	8.40	8.00	6.14	9.91	3.38	0.82	7.76
1.5KB Comb	25.49	1.71	10.46	5.14	6.03	9.52	3.15	0.92	7.80
1.5KB Comb-CS	24.66	1.76	9.06	6.90	5.82	9.72	3.32	0.73	7.75
8.0KB gshare	19.78	1.67	6.89	7.17	5.11	9.39	2.31	0.63	6.62
12KB Bi-Mode	17.65	1.67	5.97	7.51	5.17	9.23	2.34	0.59	6.27
11.75KB Comb	20.21	1.07	6.68	4.42	4.54	8.76	2.40	0.57	6.08
11.68KB Comb-CS	17.93	1.32	6.07	5.73	4.29	8.75	2.25	0.53	5.86
16.0KB gshare	16.86	1.65	6.22	6.66	4.93	9.18	2.08	0.59	6.02
24KB Bi-Mode	15.36	1.61	5.46	7.08	5.01	9.01	2.21	0.55	5.79
23KB Comb	17.68	0.96	5.81	4.41	4.45	8.64	2.30	0.49	5.59
23.5KB Comb-CS	15.15	1.18	5.47	5.36	3.99	8.58	2.05	0.49	5.29

Comb = Combining