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

2008－ARC－178（2）
2008／5／13

予測ミスした命令の実行を継続する投機手法

喜多貴信† 塩谷亮太† 入江英嗣† 五島正裕† 坂井修一†

† 東京大学大学院 情報理工学系研究科 電子情報学専攻 坂井・五島研究室
〒113-8656 東京都文京区本郷7-3-1
E-mail: †{kita,shioya,ern,goshima,sakai}@mtl.t.u-tokyo.ac.jp

あらまし プロセッサの高クロック化に伴いパイプライン段数は増加傾向にあり，予測ミスペナルティも増大している．予測ミスによってフラッシュされる命令の中には，ミスを起こした命令に依存しない命令も存在する．こうした命令をフラッシュせずに実行し，その実行結果を再利用することで，予測ミスペナルティを緩和することができる．本稿では，一度パイプラインに投入された命令は，フラッシュを行わず実行を継続するアーキテクチャを提案する．予測ミスの原因に影響されない命令の実行結果を，分岐予測や値予測のソースとして再利用することにより正しいパスの実行を支援する．シミュレータ上に提案手法を実装したところ，わずかなハードウェアの追加のみで，SPECint2000において平均3.8%（最大12%），SPECint2006において平均4.3%（最大18%），MediaBenchにおいて平均3.8%（最大36%）の性能向上を達成することができた．

キーワード 分岐予測 予測ミス

Speculation scheme that continues executing mispredicted instructions

Takanobu KITA†, Ryota SHIOYA†, Eiji IRIE†, Masahiro GOSHIMA†, and Shuichi SAKAI†

† S. Sakai and M. Goshima Lab., Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo
7-3-1 Hongo Bunkyo-ku, Tokyo 113-8656, JAPAN
E-mail: †{kita,shioya,ern,goshima,sakai}@mtl.t.u-tokyo.ac.jp

Abstract Modern processors are acquiring deeper pipelines as their clock frequencies grow higher, leading to large misprediction penalties. Not all instructions flushed on misprediction, however, are wrongly executed; there are instructions independent of control flow. Not flushing such instructions and leaving them for continued execution will reduce misprediction penalty. This paper proposes a speculation scheme which continues execution of instructions on misprediction instead of flushing them. The results of instructions independent of control flow may be used for branch prediction and value prediction, and help execution of correct path. The evaluation showed that with an addition of small amount of Hardware, the proposed scheme will increase IPCs by average of 3.8%(12SPECint2000, SPECint2006 and MediaBench.

Key words branch prediction,misprediction

1. はじめに

プロセッサの高クロック化が進むにつれて，パイプライン段数は増加する傾向にある．パイプラインが深くなると，分岐予測ミス時により多くの命令がフラッシュされることになる．近年のスーパスカラ・プロセッサでは，一度のパイプランフラッシュで50命令以上の命令がフラッシュされることもある．加えて，命令がフラッシュされた後に，命令ウィンドウに再び命令が充填されるまでにより多くのサイクル数を要することとなる．こうしたミスペナルティの増大は，プロセッサの性能低下の原因となる．

しかし，分岐予測ミス時にフラッシュされる命令の中には，分岐の成否にかかわらず実行されて，かつ同じ実行結果を生成する命令がしばしば存在する．こうした命令をフラッシュせずに実行し結果を再利用することができれば，分岐予測ミスペナルティを緩和することができる．本稿では，一度パイプラインに投入された命令はフラッシュせずに実行を継続する投機手法を提案する．予測ミスをした命令はフラッシュするかわりに，アーキテクチャ状態を変更しないように無効化しつつ実行を継続する．ミスに影響されない命令の実行結果を，分岐予測や値

```
for (col = 0; k1[col] != '\0'; col++) {
    if ( ! isgap(k1[col]) ) {
        canons1[r1] = ref[col] ? 1 : 0;
        r1++;
    }
}
```

図 1 SPECint2006 456.hmmer のコード

予測のソースとして再利用することにより正しいパスの実行を支援する．

2. では，プログラムの制御等価性とデータ非依存性について説明する．分岐合流後の共通命令の実行結果を，正しいパスの実行に利用することができれば，予測ミスペナルティを軽減することができる．3. では，制御等価な命令の実行結果を予測のソースとして利用することで，既存手法よりも大幅に単純な機構で制御等価を利用する手法を提案する．まず提案手法の具体的な流れを説明した後に，実装の詳細を解説する．4. では提案手法を評価し，長いパイプラインや大きな命令ウィンドウをもった近年のプロセッサ上における有効性を示す．

2. プログラムの制御等価性とその利用

プログラム中に含まれる分岐の多くは合流する．合流後の共通部分を利用することで，プロセッサのパフォーマンスを向上させることができる．

2.1 制御等価とデータ非依存

図 1 に，SPECint2006 の 456.hmmer のコードの一部を示す．この例で，ループ判定条件とカウンタのインクリメントは 2 行目の分岐に関わらず実行される．このような命令を**制御等価**な命令であるという．制御等価な命令のうちで，分岐の成否によって実行結果が変わる場合を**制御等価でかつデータ依存**な命令であるといい，変わらない場合を**制御等価でかつデータ非依存**な命令であるという．

2 行目の分岐，3 行目の分岐は両方とも予測ミスしやすい分岐である．要素間の値の関連性が薄い配列に対して，分岐を含むイテレーションを行うと，一般に分岐予測ミスを生じやすい．この例では，あるイテレーションにおける分岐予測ミスは次のイテレーションに影響しない．それにも関わらず，制御等価を利用しないプロセッサでは，予測ミスのたびに制御等価な命令をフラッシュしてしまう．

制御等価な命令を含むクリティカルループは，多くのプログラムに存在している．この性質を利用して，分岐予測ミス時にプロセッサ内に存在する制御等価な命令を何らかの形で正しいパスの実行に役立てることができれば，分岐予測ミスペナルティを軽減してプロセッサの性能を向上させることができる．

2.2 先行研究による制御等価の利用

Register Interation [1] は，レジスタの依存関係を解析して制御等価でかつデータ非依存な命令を識別する．物理レジスタのマッピングを変更することにより制御等価でかつデータ非依存な命令の再実行を避けるが，フェッチ幅やデコード幅は消費してしまう．Selective Branch Recovery [2] は，特定の場合 (else パートのない if 文) において，制御等価でかつデータ非依存な命令の再実行を省いた上で，さらに制御等価な命令によるフェッチ幅・デコード幅の消費を抑えている．適用可能な場面が限定される欠点がある．Skipper [3] は分岐予測ミスが起こる前に発動して制御等価を利用する．予測ミスしやすい分岐命令をフェッチすると，間をスキップして合流点のフェッチ・実行を行う．分岐先が解決した後にスキップした部分をフェッチし，あらかじめ確保しておいた命令ウィンドウにディスパッチして実行する．Ginger [4] は，前述した SBR や Wallker [5]，Skipper の特徴をそれぞれ取り入れた手法である．レジスタの再リネームを，命令ウィンドウのタグ放送ロジックを用いて行うのが特徴である．Ginger は SPECint2000 において平均 5 モデル上で Wallker と SBR，Skipper の性能評価を行っている．それによると，SPECint2000 における平均の IPC 向上率はそれぞれ 1%, 0.2%, 1%であるという．

それぞれ特徴的な方法で制御等価を利用しているが，制御等価な命令の実行結果を直接正しいパスの命令の実行結果として扱っている点で共通している．しかし，メモリを介した依存関係の正確な判定は難しく，レジスタの依存関係解析や合流点の学習なども簡単ではない．これらを行うために，先行研究では複雑なハードウェアを必要としていた．

3. 予測ミスした命令の実行を継続する投機手法

本章では，一度パイプラインに投入された命令はフラッシュを行わず実行を継続する投機手法を提案する．予測ミスをした命令はフラッシュするかわりに，アーキテクチャ状態を変更できないように無効化しつつ実行を継続する．先行研究と異なり，制御等価でかつデータ非依存な命令の実行結果は**予測のソースとして**間接的に再利用する．予測精度が十分に高ければ，直接正しいパスの命令の実行結果として扱う場合と変わらない性能を発揮できる．予測のソースとして利用する場合は予測ミスが許容されるため，複雑なハードウェアを必要としない．特に，分岐予測などの既存の予測機構のソースとして再利用する場合には，すでに備わっているミス検出・回復機構を利用することができる．

この手法は分岐予測ミスだけでなく，値予測ミスやメモリ依存予測ミスにも適用できる．先行研究の多くがレジスタ・リネーミング系のプロセッサを対象としているのに対し，提案手法は投機実行を行うプロセッサ全般に幅広く適用可能である．以下ではレジスタ・リネーミング系のスーパスカラ・プロセッサを対象モデルとし，予測ミスは主に分岐予測ミスを想定する．

3.1 必要な機構

提案手法には，「予測ミスした命令がアーキテクチャ状態を変更しないように制御する機構」と「予測ミスした命令の実行結果を利用する機構」の 2 つが必要である．

前者ついて，予測ミスした命令には **Mispredicted ビット (M ビット)** を付加することで，正しいパスの命令と区別する．M ビットのついた命令がコミットしないように制御することによりアーキテクチャ状態の変更を防ぐ．

後者について，提案手法では予測ミスした命令の実行結果を

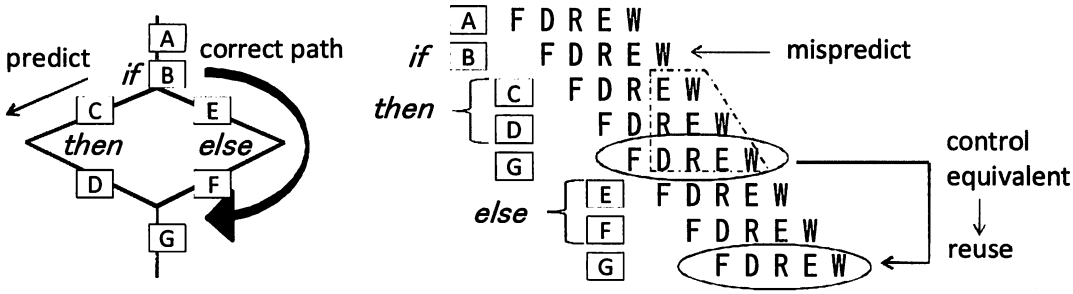

図 2　提案手法のパイプラインチャート

分岐予測のソースとして再利用するために，**Branch Result Buffer（BRB）**を実装した．BRB は，分岐命令の実行結果を格納し，図 3 のように既存の予測器のヘルパーとして機能する．

3.2　提案手法の具体的な流れ

分岐予測ミスが発生した場合，予測ミスをした命令をフラッシュするかわりに M ビットをつける．フェッチ PC やグローバル分岐履歴などを通常のプロセッサと同様に回復してプロセッサ・ステートを復元してから，再フェッチを開始する．プロセッサのパイプラインには，フラッシュによる空きスロットができることなく命令が流れていく．図 2 の例のように，通常のプロセッサではフラッシュしてしまう命令（点線で囲んだ部分）も，フラッシュせずに実行を継続する．レジスタ・マッピングテーブルは，予測ミス時にフロントエンドのパイプラインに存在していた命令をディスパッチした後に回復する．

予測ミスした命令はアーキテクチャ状態を変更してはならないので，プロセッサは M ビットのついた命令をコミットしない．ロード・ストア命令は正しいパスの命令とアドレス空間が分離され，ストアデータ・フォワーディングによって予測ミスした命令の実行結果が正しいパスの命令に渡されないように制御される．分岐命令の実行結果は BRB へ書き込まれ，正しいパス上の対応する分岐命令がフェッチされてきた時に，分岐予測のソースとして利用される．それ以外の命令については，基本的に正しいパスの命令と同様に実行する．

3.3　リソースの管理

予測ミスした命令を継続して実行するために，通常のプロセッサとやや異なった方法でリソースを管理する必要がある．以下に，主だったリソースの管理方法を説明する．

物理レジスタ

通常のプロセッサと異なり，ミス回復時に物理レジスタを解放しない．予測ミスした命令の書き込み先レジスタが，別の予測ミスした命令によって参照されている可能性があるからである．簡単のため，正しいパスの命令の実行のための物理レジスタが不足したら，予測ミスした命令群の所有していたレジスタを解放して全ての予測ミスした命令をフラッシュする．レジスタの解放速度は，デコード幅だけあれば十分である．

命令ウィンドウ

命令ウィンドウのエントリは物理レジスタとは異なり，必要に応じて解放していけばよい．単純な実装として，通常のプロセッサと同様に予測ミス回復時にエントリ番号だけをプールに返す方法がある．この場合，正しいパスの命令がディスパッチされてきたときに，予測ミスした命令のエントリに上書きしてしまうことがある．これによりプロデューサを失った命令は，リソースが不足したときにフラッシュされる．

チェックポイント

通常のプロセッサと同様に，予測ミスした命令が持っていたチェックポイントは予測ミス回復時に解放してよい．

レジスタ・マッピングテーブル

レジスタ・マッピングテーブルは予測ミス時にフロントエンドのパイプラインに存在していた命令をディスパッチした後に回復する．再フェッチされてきた命令がリネームステージに達するまで回復を遅延することにより，より多くの制御等価な命令をディスパッチして実行することができる．

アクティブ・リスト

予測ミス発覚時に，予測ミスした命令の対応するエントリは無効化される．

3.4　予測ミスした命令の実行

予測ミスした命令はストアをコミットしない．

またストアデータ・フォワーディングを行う際に，予測ミスをしたストア命令から正しいロード命令へデータをフォワーディングしない．このために，予測ミスした命令と正しい命令とでアドレス空間を分離する．通常のアドレスに M ビットを（最上位ビットなどに）付加することでこれを実現できる．

アーキテクチャ状態を変更しない命令については，基本的に通常の命令と同様に実行し物理レジスタに書き込めばよい．ただし，予測ミスした命令群に含まれる分岐命令によって分岐予測ミス回復は行わない．（後述する早期リカバリに用いる場合を除く．）

予測ミスした命令は通常はフラッシュされて実行されない命令であるため，その実行により各種予測器を汚染する可能性があることに注意する必要がある．例えば，ヒット・ミス予測器やアドレス一致・不一致予測器などの更新は問題ないが，PHT などの更新は悪影響を及ぼすことが多い．簡単のため，本稿では予測ミスした命令は一般的なプロセッサに既存の予測器の更新は一切行わないとする．

3.5　再利用可能な実行結果

予測ミスした命令の実行結果を予測のソースとして再利用す

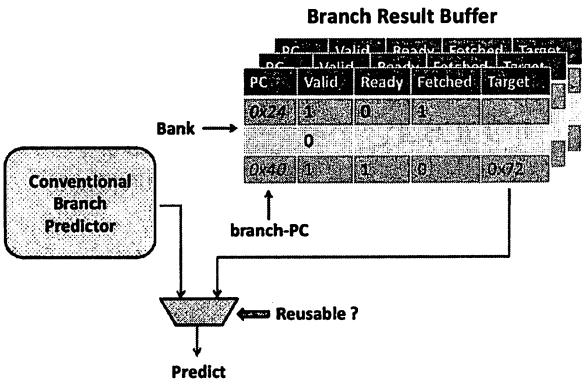

図 3 Branch Result Buffer(BRB)

るために，再利用可能性を学習・判定する必要がある．再利用可能性を高精度に判定できなければ，予測ミスペナルティによりかえって性能を下げてしまいかねない．

再利用可能な命令とは，**予測ミスの原因に依存しない命令**のことである．(正確には，予測ミスに依存していても同一の実行結果を生成する命令も含まれる.) たとえば，分岐予測ミスの場合では制御等価でかつデータ非依存な命令がこれに該当する．

予測ミスの原因に依存しない命令のみを選択的に継続実行することが理想的だが，先行研究のように分岐合流点の学習や依存関係の解析が必要となり複雑さを増す．提案手法では，予測ミスの原因に依存する命令も依存しない命令も区別せずに実行し，予測時あるいは実行終了時に予測ミスした命令の再利用可能性を判定する．選択的に継続実行をする場合と比べて余計な命令を実行することになるが，予測ミスなくパイプランが流れている時と同じ速度で実行が可能であるため，上手く設計されたプロセッサであれば実行速度に大きな影響はないと考えられる．

3.6 分岐予測のソースとしての利用

予測ミスしやすい分岐命令は，実行頻度の高い小規模なループに 1 から 4 個程度が固まって含まれていることが多い．このため，分岐予測ミス時に間違ってフェッチされた命令の中には制御等価でかつデータ非依存でかつ予測ミスしやすい分岐命令が含まれていることがある．この分岐命令の結果を回復後に利用できれば，次の分岐予測ミスの発生をおさえることができる．たとえば分岐予測ミス時に制御等価でかつデータ非依存な 3 つのクリティカルな分岐命令がパイプラインに入っていたとすれば，これらの実行結果を再利用して続く 3 度の予測ミスを回避することができる．特に短期間に複数回のミスが発生した場合には，初期のミス時にパイプラインに入っていた命令には十分な時間が与えられるため，実行終了する可能性が高くなる．

Branch Result Buffer(BRB)

Branch Result Buffer(BRB) (図 3) は，分岐命令の実行結果を格納するバッファである．BRB には予測ミスした命令かどうかに関わらず，条件分岐命令と間接分岐命令の実行結果を格納する．BRB は GShare や BTB，RAS などで構成された既存の分岐予測器にハイブリッド構成で組み込まれ，ヘルパーとして動作する．BRB の予測を採用するかどうかは，再利用可能性の判定機構によって決定する．

BRB は複数のバンクによって構成されている．バンクはフェッチ PC が後方に分岐するたびに切り替わり，ループ中の同一 PC の分岐命令を区別して扱うことができる．in-flight な命令の最大数にもよるが，4 から 32 程度で十分である．イテレーションの命令がすべてリタイアした後に，対応するバンクを解放する．各バンクは分岐命令の PC によってインデックス付けされたテーブルとなっている．クリティカル・ループは小規模なことが多いため，バンク当りのエントリ数は 4 から 32 程度が妥当であろう．エントリは分岐命令のフェッチ時に作成する．満杯の時は，バンク内でプログラム・オーダにおいて最上流の分岐命令のエントリを削除し，ここに新たなエントリを作成する．エントリは 3 つのフィールドで構成され，Valid フィールドはエントリが有効であるかどうか，Ready フィールドは対応する分岐命令の実行が終了したかどうか，Target フィールドは分岐命令の実行の結果求められた分岐先の PC，を表す．ミス回復時には後方分岐の識別用 PC とアクティブなバンク番号を復元する．

早期リカバリ

フェッチ時には BRB 中の対応するエントリが Ready でなかったために予測できなかったが，後に予測ミスした命令が実行終了した時に結果が再利用可能な場合がある．正しいパスの分岐命令がまだ実行終了していなければ，この結果を用いて分岐予測ミスを早期に検出・回復することができる．

早期リカバリを行うために，BRB のエントリに Fetched フィールドを設ける．Fetched フィールドは予測ミス回復後に，エントリに対応する分岐命令がフェッチされたかどうかを表わす．予測ミスした命令が BRB に実行結果を書き込む際に Fetched フィールドをチェックし，フラグが立っていてかつ対応する正しいパスの分岐命令の実行が終了していない場合には，早期リカバリを行う．早期リカバリを行う場合には，予測ミスからの回復時に Fetched ビットも復元する．

再利用可能な分岐命令の学習

BRB は既存の分岐予測器のヘルパーとしてはたらく．BRB が予測すべき分岐命令は，既存の予測器でのヒット率が低くさらに制御等価でかつデータ非依存な分岐命令である．

このような分岐命令を学習するために，4bit 飽和カウンタをもった学習テーブルを用いた．テーブルには，既存の予測器を用いて予測ミスをした分岐命令を記録しておく．既存の予測器が予測ミスをしたらカウンタの値を 4 アップ，ヒットしたら 1 ダウンする．カウンタの値が閾値 8 を超えた場合は，BRB を用いて分岐予測を行う．さらに，BRB を用いて予測ミスをした場合にはカウンタの値を 4 ダウン，ヒットすれば 1 アップするようにして，フィードバックをかける．テーブルの更新はリタイア時に行えばよい．ミス回数の多い分岐命令は少数であるので，学習テーブルのエントリ数は 8 から 32 程度でよい．

3.7 値予測のソースとしての利用

予測ミスした命令のうちで，制御等価でかつデータ非依存な

ものの実行結果を値予測として利用することができる．たとえば，最終値予測器やコンテキスト・ベース予測器などに応用できる．予測ミスした命令によってプリフェッチされた値を，ロード値予測に利用することも可能である．本稿では，値予測のソースとしての利用は実装しなかった．

4. 評価・考察

4.1 評価方法

提案手法を，当研究室で開発したシミュレータ「鬼斬弐」上に実装して評価を行った．命令セットは Alpha を用いた．評価には SPECint2000, SPECint2006, MediaBench を用いた．SPECint2000 から 10 本，SPECint2006 から 9 本を用い，最初の 1G 命令をスキップしたのち続く 100M 命令を実行した．入力には train を用いた．MediaBench からは 16 本を用い，プログラム終了までの全命令を実行した．ベースモデル（表 1）として，大きな命令ウィンドウと深いパイプラインをもったプロセッサを想定した．分岐予測ミスの回復は register read ステージで行うため，ミスペナルティは 15 サイクルである．命令キャッシュのアライメントは考慮していない．メモリ依存予測は楽観的に行い，アクセスオーダ・バイオレーションやレイテンシ予測ミスの回復は選択的再スケジューリングによって行った．予測ミスした命令の実行優先度は低く設定した．BRB のサイズは 16 バンクでバンク当り 16 エントリ，再利用可能性を学習するためのテーブルのサイズは 32 エントリとした．

表 1 ベースモデル

Pipeline	20 stages: 1 predict, 3 fetch, 4 rename, 2 dispatch, 2 issue, 3 register read, 1 execute, 3 register write, 1 retire. 15 minimum branch misprediction penalty.
Fetch	4 instructions (up to 1 taken branch)
Issue	int:4, fp:2, mem:2
Isuue queue	int:128, fp:64, mem:64
Register file	int:256, fp:128
Memory hierachy	64KB, 4-way associative, 3-cycle instruction and data caches. 4MB, 8-way associative, 12-cycle L2. 150-cycle main memory.
Branch predictor	10-branch history 32K-entry gShare. 4K-entry 4-way associative BTB. 32-entry RAS.
BRB	16-bank, 16-entry per bank. 32-entry reusability table.

4.2 評価結果

図 4 に評価結果を示す．バーは左から順に，ベースモデルと提案手法の分岐予測ヒット率（最大値が 1）と，相対 IPC の関係を表わしている．右端に，すべてのベンチマークの平均を表わしている．

4.2.1 分岐予測ヒット率の向上

SPECint2000, SPECint2000, MediaBench における分岐予測ヒット率向上の平均は 0.63%, 0.64%, 0.95%であった．

ベンチマークによってヒット率向上が大きいものと小さいものとに二極化している．これは，クリティカル・ループ中にミス率が高くて制御等価でかつデータ非依存な分岐命令が含まれるかどうかに依存していると考えられる．そのような分岐命令が含まれている場合は，1 度ミスが起きたのちに続く数個のクリティカルな分岐命令がヒットするため，ヒット率を大きく向上させることができる．(456.hmmer や adpcm.d など）逆に，254.gap や 255.vortex においては分岐予測ヒット率が下がっている．これは BRB が再利用可能性を正しく判定できず，予測ミスしてしまったことが主な原因であると考えれらる．もともとの分岐予測ヒット率が高いベンチマークにこの傾向が強い．

4.2.2 IPC の向上

SPECint2000, SPECint2000, MediaBench における IPC 向上率の平均は 3.81%, 4.33%, 3.77%であった．

adpcm.d や 456.hmmer は非常に大きな性能向上を達成している．これらのベンチマークは，キャッシュヒット率が非常に高く分岐予測ヒット率が低いために，分岐予測ミスペナルティによる性能低下の割合が大きい．そのため分岐予測ヒット率の大幅な向上によって，IPC も大きく向上させることができた．181.mcf も比較的大きな IPC 向上を示している．このベンチマークは，キャッシュミス率，分岐予測ミス率ともに非常に高い．同時に，ミス率の高いロード命令や分岐命令が，小さなクリティカル・ループに集中的に存在していることも特徴である．この特徴により，予測ミスした命令を継続実行した際に多くのクリティカルなロード命令や分岐命令を実行できたことが，キャッシュヒット率向上や分岐予測ヒット率向上につながったと考えられる．ほとんど性能向上のなかったり，IPC がわずかに下がっているベンチマークもある．

4.3 考　　察

IPC や分岐予測ヒット率の向上がほとんどないベンチマークがある．これらは，もともと分岐予測ヒット率が非常に高かったり，プログラム中のクリティカルなコード中に制御等価性が少なかったり（あっても合流点までの距離が長すぎて制御等価を利用できない）することが原因であろう．

ベンチマークの中には，IPC がわずかに下がっているものがあった．IPC が下がる原因としては，おもに以下の理由が考えられる．

- BRB の分岐予測ミス
- 予測ミスした命令によってキャッシュが汚染された
- 予測ミスした命令によって命令の発行幅が消費された

BRB の分岐予測ミスについては，一部のベンチマークでは多発している．これは，より洗練された方法で再利用可能性を判定する必要があることを意味している．キャッシュの汚染については，それほど深刻ではなくむしろわずかにヒット率が上がっているものが多かった．予測ミスした命令が発行幅を消費してしまい，正しいパスの実行が遅れてしまう可能性もある．これらに関しては，マイナスに作用する場合には予測ミスした命令の継続実行や BRB による予測を行わないように制御することで改善することができる．BRB の再利用可能性を学習するテーブルも，BRB の予測がはずれた場合にフィードバックをかけて，BRB による予測を抑制する機能がついているが，より高精度なものが必要とされる．マイナスの作用を抑制するチュー

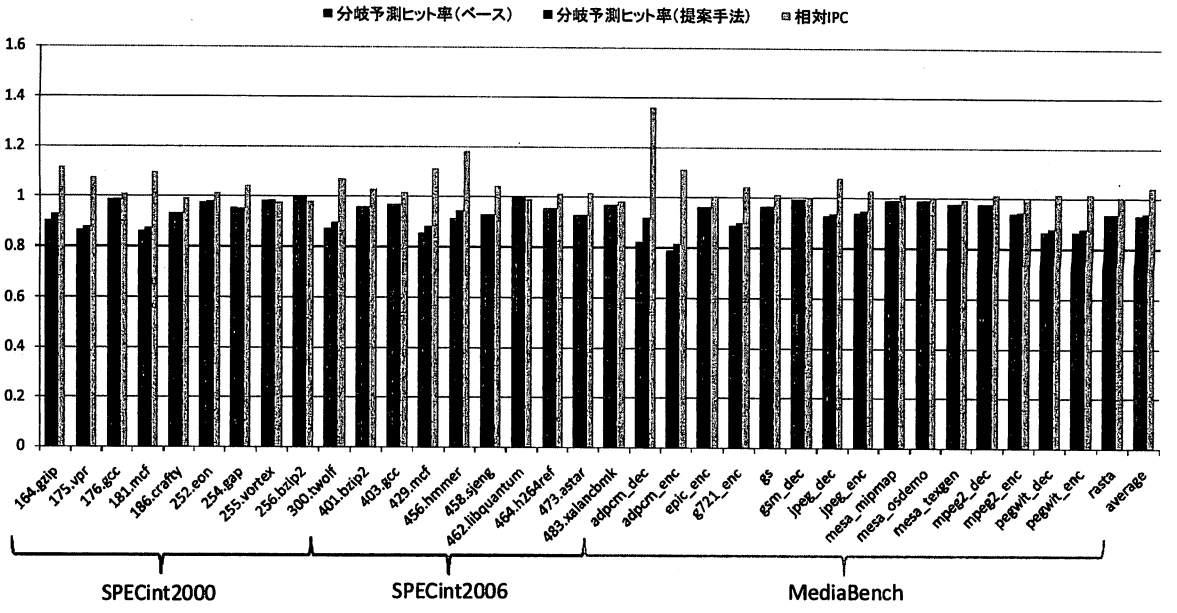

図 4　分岐予測ヒット率と相対 IPC の関係

ニングにより，さらなる性能向上を期待できる．

35 種という多様なベンチマークを評価することによって，提案手法が非常に効果的に作用するプログラムが存在することがわかった．効果の薄いベンチマークに対しても，提案手法が性能低下を引き起こすことはない（あってもわずかで，改善も可能）．

5. ま と め

本稿では，予測ミスした命令の実行を継続する投機手法を提案した．予測ミスした命令の実行結果は予測器のソースとして再利用され，予測ミス回復後の命令実行に寄与する．予測ミスした命令の実行結果を，予測ミス回復後の命令の結果として直接的に利用するのではなく，予測のソースとして利用することで，ハードウェアを大幅に単純化することができる．適用可能なアーキテクチャを選ばないことも特徴である．

プロセッサ・シミュレータ上に提案手法を実装し，わずかなハードウェアの追加のみで，平均約 4%の有意な性能向上を達成することができた．特に一部のベンチマークには非常に効果的で 36%もの性能向上を示すものもあった．パイプラインの深化や命令ウィンドウの大型化によって in-flight な命令数が増えつつある近年のプロセッサにおいて，提案手法の有効性を示すことができた．

謝辞　本論文の研究は，文部科学省科学研究費補助金（基盤研究（B）　No.20300015）による．

文　献

[1] Roth, A. and Sohi, G. S.: Register Integration: A Simple and Efficient Implementation of Squash Reuse, *Int'l Symp. on Microarchitecture (MICRO)*, pp. 223–234 (2000).

[2] Gandhi, A., Akkary, H. and Srinivasan, S. T.: Reducing Branch Misprediction Penalty via Selective Branch Recovery, *Int'l Conf. on High-Performance Computer Architecture (HPCA)*, pp. 254–264 (2004).

[3] Cher, C.-Y. and Vijaykumar, T. N.: Skipper: A Microarchitecture for Exploiting Control-flow independence, *Int'l Symp. on Microarchitecture (MICRO)*, pp. 4–15 (2001).

[4] Hilton, A. D. and Roth, A.: Ginger: Control Independence Using Tag Rewriting, *Int'l Symp. on Computer Architecture (ISCA)*, pp. 436–447 (2007).

[5] Sodani, A. and Sohi, G. S.: Dynamic Instruction Reuse, *Int'l Symp. on Computer Architecture (ISCA)*, pp. 194–205 (1997).