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

2006－ARC－167（1）
2006－HPC－105（1）
2006／2／27

クリティカル・パス上の命令に着目したレジスタ・キャッシュの使用法

小林　良太郎†　梶山　太郎†　島田　俊夫†

近年のハイエンドプロセッサにおいて動作周波数の向上を制限する要因の一つであるレジスタ・ファイルへのアクセス時間を改善する手法として階層型レジスタ・ファイルがある．この機構は高速に動作するレジスタ・キャッシュを持ち，そのミス率がプロセッサ性能に大きな影響を及ぼす．本論文において，我々はプログラム全体の実行時間を決定するクリティカル・パスに着目し、レジスタ・キャッシュ・ミスが発生してもプロセッサ性能が低下しにくい方式を提案する．

Usage of Register Cache Focusing on Critical Path

Ryotaro Kobayashi,† Taro Kajiyama† and Toshio Shimada†

The hierarchical register file has the potential to improve improving the access time of the register file that is one of the factors to limit the improvement of the clock frequency in recent high-end microprocessor. As for this mechanism, it has the register cache that operates with a faster clock rate, and the miss rate has a great influence on the processor performance. In this paper, we focus on a critical path in a program and propose a mechanism that alleviates performance degradation due to register cache misses.

1. はじめに

近年のハイエンド・プロセッサは，複数命令発行によって命令レベル並列性を利用し，パイプライン段数を深くすることによって動作周波数の向上を実現している．しかし，命令レベル並列性を利用するために，レジスタ・ファイルは多くのエントリとポートを必要とする．そのため，レジスタ・ファイルのアクセス時間は増加し，動作周波数の向上を妨げる要素の１つとなっている[6]．

この問題を解決する方法の１つとして，階層型レジスタ・ファイルが提案されている[1)4)9)]．この機構では，レジスタ・ファイルを階層型し，小容量で高速な**レジスタ・キャッシュ（RC: Register Cache）**と，大容量で低速なメイン・レジスタ・ファイル（MRF: Main Register File）で構成する．MRF は全てのレジスタ値を保持し，RC にはその一部を保持する．高速にアクセス可能な RC を用いることで，レジスタ・ファイルのアクセス時間が動作周波数に与える影響を減少させることができる．しかし，RC は一部のレジスタ値しか保持しないので，RC へのアクセスがミスする場合がある．この場合，レジスタ・ファイルのアクセス・レイテンシが増加し，プログラムの実行サイクル数に悪影響を及ぼすという問題がある．これに加え，ミスした命令に依存する命令が誤って発行されることにより，他の命令の発行が妨げられるという問題がある．

したがって，階層型レジスタ・ファイルでは，RC をどのように更新するかが重要となる．これまでに提案された方式は，RC ミスの頻度を減少させることを目的とするものである[1)4)9)]．しかし現実には，ミスの発生を完全に無くすことはできないので，プロセッサ性能への悪影響を十分に緩和することができないという問題がある．

この問題に対して我々は，プログラム全体の実行時間を決定するクリティカル・パス[3)5)8)] に着目し，ミスが発生しても，プロセッサ性能が低下しない方式を提案する．具体的には，まず，クリティカル・パス上の命令がミスを起こさないように RC の更新を行う．さらに，RC の更新情報に基づいて，ミスする命令を検出し，それに依存する命令が誤って発行されるのを防ぐ．

以下，2 章では階層型レジスタ・ファイルについて述べる．3 章で提案手法について述べ，4 章で提案機構について述べる．5 章で評価を行う．最後に 6 章でまとめる．

2. 階層型レジスタ・ファイル

図 1 に階層型レジスタ・ファイルの構成を示す．図においてデータパスは，機能ユニット (EU: Execute Unit)，階層型レジスタ・ファイルを持つ．階層型レジ

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

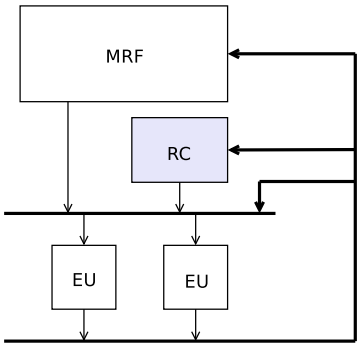

図 1　階層型レジスタ・ファイル

スタ・ファイルは，RC と MRF で構成される．MRF は大容量かつ低速なレジスタ・ファイルであり，全てのレジスタ値を保持する．一方，RC は小容量かつ高速なレジスタ・ファイルであり，一部のレジスタ値のみ保持する．高速な RC を用いることで，レジスタ・ファイルのアクセス時間が動作周波数に与える影響を緩和することができる．

発行された命令は，オペランド・バイパス，あるいは，階層型レジスタ・ファイルへのアクセスによってオペランドを得る．階層型レジスタ・ファイルにアクセスする場合，まず最初に，RC にアクセスする．RC は一部のレジスタ値しか保持しないので，アクセスがヒットする場合とミスする場合の 2 通り存在する．RC がヒットした場合，高速な RC からオペランドを得ることができる．一方，RC がミスした場合，低速な MRF からレジスタ値を得なければならない．この場合，レジスタ・アクセス・レイテンシが増加し，実行サイクル数に悪影響を及ぼす．

この問題に加え，RC ミスの発生が他の命令の発行を妨げるという問題がある．ここで，RC ミスする命令 `i0` とその結果を使用する命令 `i1` を考える．パイプライン構成は，フェッチ 2 段，デコード 2 段，発行 1 段，オペランド・アクセス 1 段，実行 1 段，ライトバック 1 段，コミット 1 段とする．RC 上にはまだどの値も書き込まれていないとする．

あるサイクルにおいて `i0` が発行されると，その次のサイクルにおいて `i0` が RC にアクセスする．また，このアクセスがヒットするとして，`i1` が投機的に発行される．しかし実際には，`i0` は RC ミスするので，その次のサイクルにおいて，`i1` はオペランド・アクセスができない．したがって，`i1` の発行は誤っていたことが分かる．このように投機的発行に失敗した命令はプロセッサの限られた発行幅を浪費するので，他に有効な命令が存在した場合，それらの発行を妨げてしまうことになる．

これらの問題があるため，階層型レジスタ・ファイルでは，RC の更新方法が重要となる．RC の更新方法は大きく 2 つに分類できる．1 つは，命令の生成した値のうち，どの値を RC に書き込むかを決める**書き込みポリシー**である．もう 1 つは，書き込みポリシーによって選択した値を，RC のどのエントリに書き込むかを決める**置き換えポリシー**である．これまでに幾つかの書き込みポリシーや置き換えポリシーが提案されてきた[1)4)9)]．これらの研究では，RC ミスの頻度を減少させることを目的としている．例えば，Non Bypass 方式では，オペランド・バイパスしなかった値のみ RC に書き込む．これにより，RC への書き込み頻度を減らし，必要な値が RC から追い出されることを防ぐことができる．

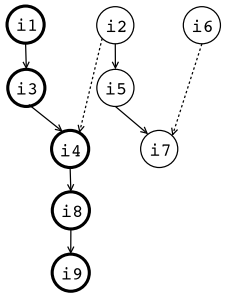

図 2　DFG

しかし，RC は一部のレジスタ値しか保持しないため，現実には RC ミスを完全に無くすことはできない．したがって，RC ミスの削減しか考慮しない従来手法では，RC ミスがプロセッサ性能に与える悪影響を十分に緩和することができない．

3. クリティカル・パス情報を利用した RC の使用法

RC ミスがプロセッサ性能に与える悪影響を緩和するため，我々は，プログラム中のクリティカル・パスに着目する．クリティカル・パスとは，プログラムの実行サイクル数を決定する命令列である．クリティカル・パス上の命令の実行が遅れると，プログラム全体の実行サイクル数が増加する．**図 2** に，クリティカル・パスの例を示す．図において，ノードは命令を示す．エッジはデータ依存関係を示し，実線がオペランド・バイパスによって，点線がレジスタ・ファイル・アクセスによって値が渡されたことを示す．`i`n は命令につけられた名前であり，n は命令のフェッチ順を示す．強調された ノード `i1`，`i3`，`i4`，`i8`，`i9` で構成されるパスはクリティカル・パスであり，データ・フロー・グラフ (DFG: Data Flow Graph) における最も長いパスとなる．

まず，RC ミスによるレジスタ・アクセス・レイテンシ増加の影響を緩和する方策を考える．

クリティカル・パス上の命令に RC ミスが発生すると，当該命令のレジスタ・アクセス・レイテンシが増加するため，プログラム全体の実行サイクル数が増加してしまう．そこで我々は，クリティカル・パス上の命令に RC ミスを発生させない手法を提案する．これにより，RC ミスが発生したとしても，性能に与える悪影響を緩和することができる．

目的を実現する手法として，クリティカル・パス上

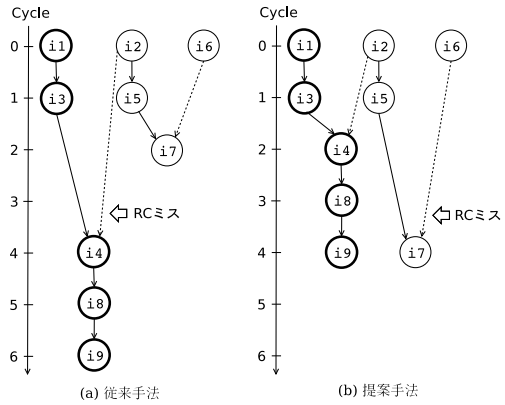

図 3 RC ミスの影響の緩和

の命令が使用する値を全て RC に書き込むという方法が考えられる．しかし，オペランド・バイパスによって得られる値は RC に書き込む必要が無いだけでなく，オペランド・バイパスでは得られない値を RC から追い出してしまう可能性がある．そこで実際には，クリティカル・パス上の命令が使用する値の中で，オペランド・バイパスにより得られない値のみを RC に書き込むこととする．以降の説明では，クリティカル・パス上の命令が使用する値を **C-Data**（Critical Data）と呼ぶこととし，C-Data のうちオペランド・バイパスでは得られない値を **NBC-Data**（Non-Bypassed Critical Data）と呼ぶこととする．

図 3 を用いて，提案手法の目的とする動作を説明する．この図は，図 2 の命令列を実行する時の過程である．縦軸は命令が実行されるクロック・サイクルを示す．ノードは命令を示し，エッジはデータ依存関係を示す．実線のエッジがオペランド・バイパスによって，点線のエッジがレジスタ・ファイル・アクセスによって値が渡されたことを示す．RC のミス・ペナルティは 2 サイクルとする．説明を容易にするため，RC のエントリ数は 2 とする．

図 3(a) は，従来手法の場合を示す．従来手法では RC ミスを削減することしか考慮していない．そのため，図に示すように，クリティカル・パス上の `i4` に RC ミスが発生することを防ぐことができない．`i4` は，RC ミスによりペナルティ（2 サイクル）が課せられ，実行が 4 サイクル目まで遅れる．その結果，プログラム全体の実行サイクル数が増加してしまう．

一方，図 3(b) は提案方式の場合を示す．図に示すように，我々の方式では，クリティカルパス上の命令に RC ミスが発生しないように RC の更新を行う．具体的には，C-Data（`i1` 〜`i3`，`i4`，`i8` の結果）を全て RC に書き込むのではなく，NBC-Data（`i2` の結果）のみを RC に書き込む．その結果，非クリティカル・パス上の `i7` は RC がミスするが，クリティカル・パス上の `i4` は RC がヒットするので，RC ミスによる実行サイクル数の増加を防ぐことができる．

次に，RC ミスの発生が他の命令の発行を妨げるという問題に対応する方策を考える．

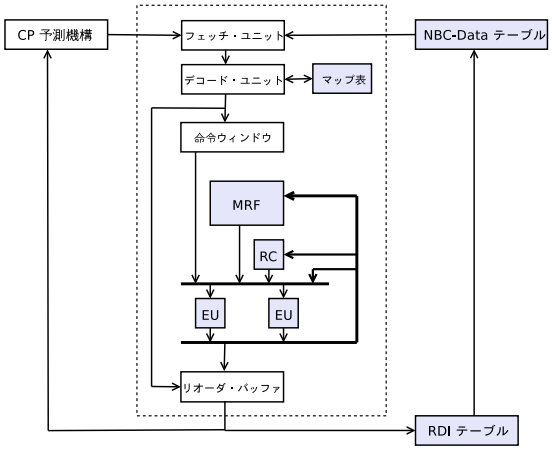

図 4 提案機構の全体図

この問題の原因は，2 章で説明したように，RC ミスする命令に依存する命令を，投機的に発行してしまうことにある．そこで，RC ミスする命令を事前に検出し，それに依存する命令は投機的に発行しないという手法を提案する．RC ミスする命令を事前に検出するため，先程我々が提案した手法を利用する．この手法では，NBC-Data は RC に書き込むが，それ以外の値は一度も RC に書き込まれることがない．したがって，オペランドが NBC-Data でない命令は，必ず RC ミスすることが分かる．

例えば，図 3(b) に示すように，提案方式では，NBC-Data である `i2` の結果のみ RC に書き込む．したがって，オペランドが NBC-Data ではない `i7` は，必ず RC ミスすることが分かる．

これらをまとめると，提案方式では，RC ミスの影響を緩和するために，以下 2 つの動作を行う．

(1) NBC-Data を RC に書き込む．
(2) RC ミスする命令に依存する命令を投機的に発行しない．

4. 提 案 機 構

4.1 提案機構の概要

提案機構を備えたプロセッサの例を**図 4** に示す．図中央の点線で囲まれた部分は，階層型レジスタ・ファイルを搭載したプロセッサで，フェッチ・ユニット，デコード・ユニット，命令ウインドウ，RC，MRF，機能ユニット，リオーダ・バッファを持っている．

プロセッサの左側は，クリティカル・パス予測機構である．この機構を用いて，フェッチした命令がクリティカル・パス上にあるかどうかを判断する．

提案機構は大きく以下 2 つの機構に分けることができる．

(1) NBC-Data 予測機構
(2) RC ミス検出機構

NBC-Data 予測機構は，プロセッサ右側の **RDI テーブル（Register Definition Information Ta-**

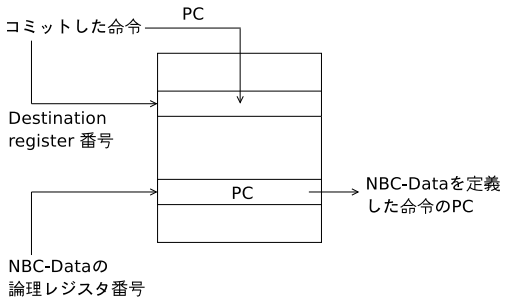

図 5　RDI テーブル

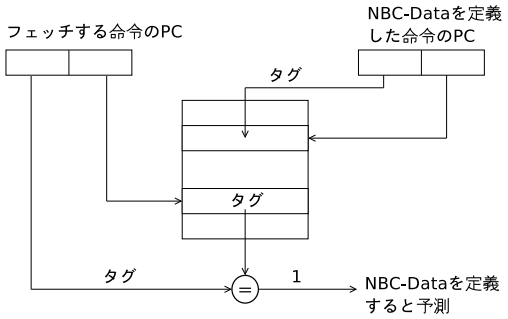

図 6　NBC-Data テーブル

ble）と **NBC-Data テーブル（Non-Bypassed Critical Data Table**）で構成する．

NBC-Data 予測機構は，実行結果が NBC-Data となる命令を予測し，その命令の実行結果のみを RC に書き込む．こうして，RC ミスによるレジスタ・アクセス・レイテンシの増加を防ぐ．

一方，RC ミス検出機構はマップ表に簡単な修正を加えるだけで実現できる．RC ミス検出機構は，NBC-Data をオペランドとしない命令を，RC ミスする命令であると検出する．そして，RC ミスすると検出した命令に依存する命令に対して投機的発行が行われないようにする．これにより，RC ミスが発生しても，それに依存する命令が他の命令の発行を妨げることがなくなる．

以降の節では，これらの機構の詳細について述べる．

4.2　NBC-Data 予測機構

NBC-Data 予測機構では，過去に実行結果が NBC-Data であった命令は，次回実行するときにも実行結果が NBC-Data となると予測する．具体的には，RDI テーブルを用いて，コミットした命令のうち，実行結果が NBC-Data であった命令の PC を求める．その結果得られた PC を，NBC-Data テーブルに登録する．命令フェッチ時に NBC-Data テーブルを参照し，このテーブルに PC が登録されている命令は，実行結果が再び NBC-Data になると予測する．

以下の節では，RDI テーブルと NBC-Data テーブルの詳細について説明する．まず，4.2.1 項と 4.2.2 項では，説明を容易にするために，命令間のデータ依存関係は実行中に変化しないと想定して，これらの機構を提案する．次に 4.2.3 項において命令間のデータ依存関係が動的に変化した場合にも対応できるように，機構を修正する．

4.2.1　RDI テーブル

実行結果が NBC-Data であった命令の PC を求めるために，**図 5** に示す RDI テーブルを用いる．RDI テーブルは，論理レジスタ番号をインデクスとする．各エントリは，対応するレジスタを定義した命令の PC を保持する．テーブルの更新は，コミットした命令のデスティネーション・レジスタに対応するエントリに，その命令の PC を書き込むことで行う．

クリティカル・パス上の命令をコミットする時に，オペランド・バイパスでは得られなかったオペランド（NBC-Data）があれば，その NBC-Data の論理レジスタ番号で RDI テーブルを参照する．参照した結果得られる PC が，NBC-Data を定義した命令の PC である．

4.2.2　NBC-Data テーブル

NBC-Data テーブルの構成を**図 6** に示す．NBC-Data テーブルは，PC の下位ビットをインデクスとする．各エントリは，NBC-Data を定義した命令の PC を識別するためのタグを保持する．命令の PC のうち，インデクスとして用いない上位ビットをタグとして用いる．

RDI テーブルを用いて NBC-Data を定義した命令の PC が求められると，その PC を識別するタグを NBC-Data テーブルに登録する．具体的には，NBC-Data テーブルにおいて，PC の下位ビットに対応するエントリに，残りの上位ビットをタグとして書き込む．

命令フェッチ時に PC の下位ビットをインデクスとして NBC-Data テーブルを参照し，得られたタグと，フェッチする命令のタグを比較する．両者が一致すれば，フェッチする命令の実行結果は NBC-Data になると予測する．

4.2.3　実行結果が NBC-Data でなくなった命令への対応

前項までの説明では，命令間のデータ依存関係は実行中に変化しないと想定し，NBC-Data テーブルに登録された命令は，それ以降常に，実行結果が NBC-Data になると予測していた．しかし実際には，命令間のデータ依存関係は実行中に変化する．NBC-Data テーブルに登録された命令の実行結果が，NBC-Data にならない場合も存在する．その場合，NBC-Data の予測は失敗するので，NBC-Data テーブルの登録内容を変更する必要がある．

そこで，以下のように機構を修正する．まず，RDI テーブルの各エントリに，NBC-Data の予測が失敗したかどうかを調べるためのフラグを追加する．このフラグを **P(Predicted) ビット**と呼ぶ．P ビットは，対応するエントリが NBC-Data と予測されたとき，1 にセットし，その予測が当たっていたとき，0 にリセットする．したがって，P ビットが 1 のままであったエントリは，NBC-Data の予測に失敗したことが分かる．

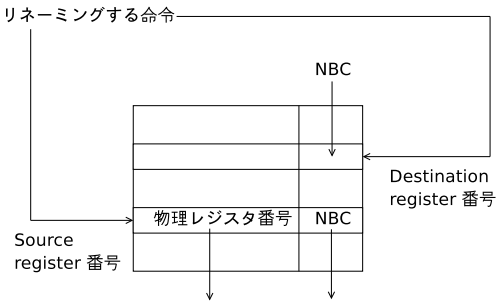

図 7　修正を加えたマップ表

まず，P ビットの更新動作を示す．コミットした命令は，実行結果が NBC-Data であると予測されていた場合，デスティネーション・レジスタに対応する P ビットを 1 にセットし，そうでない場合 0 にリセットする．また，クリティカル・パス上の命令であると予測されていた場合，オペランド・バイパスによって得られなかったオペランド（NBC-Data）があれば，NBC-Data の論理レジスタ番号に対応する P ビットを 0 にリセットする．

次に，NBC-Data 予測の失敗した命令の PC を，NBC-Data テーブルから削除する動作を示す．コミットした命令は，まず，RDI テーブルのデスティネーション・レジスタに対応するエントリを参照し，これから上書きするエントリの情報を調べる．P ビットが 1 の場合，そのエントリは NBC-Data の予測に失敗していたことを示しているので，上書き前の PC をインデクスとして NBC-Data テーブルを参照し，登録されているタグをクリアする．

4.3　RC ミス検出機構

提案する手法では，NBC-Data のみ RC に書き込む．RC ミス検出機構は，この性質を利用して，オペランドが NBC-Data でない命令を，RC ミスする命令であると検出する．

オペランドが NBC-Data かどうかを求めるために，マップ表を修正する．通常，マップ表の各エントリは，対応する論理レジスタに割り当てられた物理レジスタ番号を保持する．修正を加えたマップ表では，各エントリに対し，対応する論理レジスタが NBC-Data であることを示すフラグを追加する．このフラグを **NBC ビット**と呼ぶ．NBC ビットが 1 ならば NBC-Data であり，0 ならば NBC-Data ではない．

図 7 に修正を加えたマップ表を示す．リネーミング時に，まず命令のソース・レジスタ番号でマップ表を参照する．NBC ビットが 0 の場合，オペランドは NBC-Data ではないので，当該命令を RC ミスする命令と検出する．そうでない場合，RC ヒットと検出する．次に，当該命令の実行結果が NBC-Data となると予測されている場合，デスティネーション・レジスタに対応する NBC ビットを 1 にセットし，そうでない場合，0 にリセットする．

例えば，図 2 の命令列において，`i6` の実行結果は NBC-Data ではないと予測された場合，デスティネーション・レジスタに対応する NBC ビットを 0 にリセットする．`i7` は，この NBC ビットを参照することによって，`i6` から受け取る値が NBC-Data でないことが分かる．これにより，`i7` は RC ミスする命令であると検出することができる．

表 1　測定条件

フェッチ幅	8 命令
発行幅	8 命令
命令ウィンドウ	128 エントリ
ROB	256 エントリ
LSQ	64 エントリ
RC	ミスペナルティ 2 サイクル
MRF	int, fp 各 256 個
機能ユニット	iALU 8, iMULT/DIV 2, fpALU 4, fpMULT/DIV/SQRT 2
命令キャッシュ	完全，ヒットレイテンシ 1 サイクル
データキャッシュ	32KB, 2 ウェイ，32B ライン，4 ポート，ミスペナルティ6 サイクル
2 次キャッシュ	2MB, 4 ウェイ，64B ライン，ミスペナルティ36 サイクル
ストアセット	8K エントリ SSIT, 4K エントリ LFST
分岐予測機構	BTB 2048 エントリ，4 ウェイ，gshare 6 ビット履歴 8K エントリ PHT，予測ミスペナルティ6 サイクル
CP 予測器	24 トークン, 予測テーブル 64K エントリ

5. 評　　価

5.1　評 価 環 境

シミュレータには，SimleScalar Tool Set[2] のスーパースカラ・プロセッサ用シミュレータを用い，階層型レジスタ・ファイルと提案機構を組み込んで評価した．命令セットには MIPS R10000[7] を拡張した SimpleScalar/PISA を用いた．ベンチマーク・プログラムは，SPECint2000 の bzip2，gcc，gzip，mcf，paser，perl，votex，vpr の 8 本を使用した．gcc では 1G 命令，その他では 2G を命令スキップした後，100M 命令を実行した．測定条件として**表 1** を用いた．クリティカル・パス予測器として Token 伝搬型予測器[5] を用いた．

以下のモデルに対して評価を行った．

- **AC**(All Cache) モデル: 命令の生成した値を全て RC に書き込むモデルである．
- **NB**(Non Bypass) モデル: オペランド・バイパスによって後続命令に渡さなかった値のみ RC に書き込むモデルである．
- **CP**(Critical Path) モデル: 提案手法を用いたモデルである．
- **ideal** モデル: RC が全てヒットするとした理想的なモデルである．

どのモデルにおいても，置き換えポリシーとして，最も最近使用されていないエントリにレジスタ値を書き込む LRU 方式を用いた．

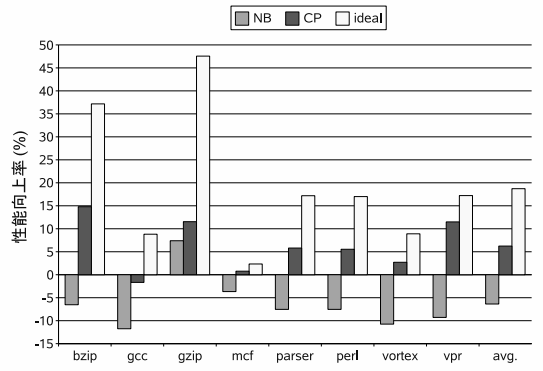

図 8　性能向上率（エントリ数 32）

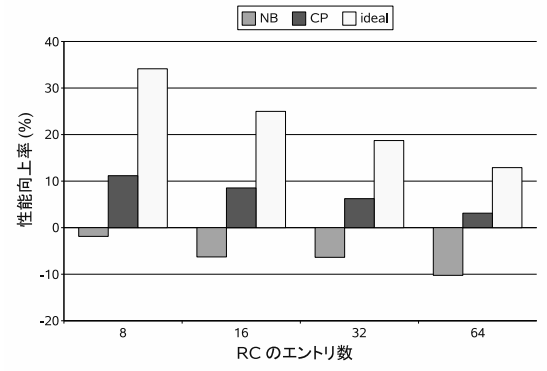

図 9　エントリ数と性能向上率の関係

5.2 測定結果

図 8 に，RC エントリ数を，論理レジスタ数と同じ 32 としたときの性能向上率を示す．図の縦軸は AC モデルに対する IPC の向上率を示し，横軸はベンチマークを示す．3 本で組になっている棒グラフは，左から順に NB モデル，CP モデル，ideal モデルである．

図から分かるように，NB モデルはどのベンチマークにおいても性能が低下することが分かる．AC モデルに対する性能低下率は，最大 11.8%，平均で 6.4%となる．NB モデルでは，オペランド・バイパスを行った値は後続命令に渡した値は RC に書き込まない．これは，命令の結果値は 1 度しか参照されないと想定しているからである．しかし実際には，命令の結果値は 2 度以上参照される場合が少なからず存在するため，NB モデルは性能が低下すると考えられる．

一方，提案機構である CP モデルは，AC モデルに対して，最大 14.8%，平均で 6.3%の性能向上を達成している．また，NB モデルに対しては，最大 22.9%，平均 13.5%の性能向上を達成している．このことからクリティカル・パスに着目した効果が現れていると考えられる．

図 9 に，RC エントリ数を 8, 16, 32, 64 と変化させたときの性能向上率を示す．図の縦軸は AC モデルに対する性能向上率を示し，横軸は RC エントリ数を示す．3 本で組になっている棒グラフは，左から順に NB モデル，CP モデル，ideal モデルである．

図から分かるように，CP モデルは，エントリ数が少ないほど，性能向上率が高くなることがが分かる．AC モデルに対する CP モデルの性能向上率は，エントリ数が減少するにつれて，3.1%，6.3%，8.5%，11.2%と増加して行く．この理由は，提案手法が NBC-Data しか RC に書き込まないので，エントリ数が減少しても，RC 上での競合はそれほど増加しないことにある．

6. まとめ

本論文では，プログラム全体の実行時間を決定するクリティカル・パスに着目し，ミスが発生しても，プロセッサ性能が低下しない手法を提案した．提案手法では，クリティカル・パス上の命令がミスを起こさないように RC を更新し，投機的発行の失敗を避けるために RC ミスの検出を行う．

評価の結果，提案手法は，RC のエントリ数が 32 のとき，全ての結果値を RC に書き込む All Cache 方式を用いたモデルに対して，IPC で最大 14.8%，平均で 6.3%性能が向上することが分かった．また，提案手法はエントリ数の少ない RC において高い性能を得られることが分かった．

謝辞　本研究の一部は，文部科学省科学研究費補助金基盤研究（C）課題番号 15500036，文部科学省 21 世紀 COE プログラムの支援により行った．

参考文献

[1] J. A. Butts, et al., “Use-Based Register Caching with Decoupled Indexing,” In *Proc. ISCA-31*, March 2004.

[2] D. Buger, et al., “The SimpleScalar Tool Set, Version 2.0,” Technical Report CS-TR-97-1342, June 1997.

[3] 千代延昭宏ほか: 低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案, 情報処理学会研究報告 2002-ARC-149, 2002 年 8 月.

[4] J. L. Cruz, et al., “Multiple-Banked Register File Architectures,” In *Proc. ISCA-27*, June 2000.

[5] B. Fields, et al., “Focusing Processor Policies via Critical-Path Prediction,” In *Proc. ISCA-28*, June 2001.

[6] R. Gonzalez, et al., “A Content Aware Integer Register File Organization,” In *Proc. ISCA-31*, March 2004.

[7] MIPS Technologies, Inc., “Combining Branch Predictors,” WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.

[8] E. Tune, et al., “Dynamic Prediction of Critical Path Instruction,” In *Proc. HPCA-7*, January 2001.

[9] R. Yung, et al. “Caching Processor General Registers,” In *Proc. ICCD*, October 1995.