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

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

リネーミングされるレジスタ番号の整列による レジスタ・キャッシュの高精度化手法

小林　良太郎†　堀部　大介†　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，そのヒット率が性能に大きな影響を及ぼす．本論文では，物理レジスタ番号の割当順に着目し，モジュロ・ソーティング・バッファと呼ぶ機構を導入することで，レジスタ・キャッシュの構成を複雑化することなく，ヒット率を向上させ，プロセッサの性能を向上する手法を提案する．

High Accuracy of Register Cache with Ordering of Physical Register Number

Ryotaro Kobayashi,† Daisuke Horibe† and Toshio Shimada†

The hierarchical register file has the potential to improve 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 the allocation order of physical register number, and propose Modulo Sorting Buffer to improve the hit rate of register cache without increase of design complexity, improving performance.

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・ファイル機構が提案されている[1),4),5),9)]．この機構では，レジスタ・ファイルを階層化し，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緩和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・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．したがって，階層型レジスタ・ファイルでは，RC ミスをどのように削減するかが重要となる．

RC ミスを削減するアプローチの１つに，実行ユニットから新たに得られたレジスタ値を，参照される可能性の低いエントリに書き込むというものがある．これを実現するため，いくつかの方式が導入されている[4),9)]．しかしそれらの多くは，値を RC へ書き込む直前に，書き込み先のエントリを決めており，処理が複雑となる．さらに，どのエントリに参照すべきレジスタ値があるのかを知るため，RC の連想検索を行う必要がある．そのため，RC のアクセス時間が増加してしまうという問題がある．

RC のハードウェアを最も簡単にできるものとして，**ダイレクト・マップ**方式を挙げることができる．この方式では，レジスタ・リネーミング時に決まる物理レジスタ番号だけをもとに，読み書きするエントリを決めるので，RC が非常に簡単に実現できる．そこで我々は，RC をダイレクト・マップ方式とし，データパスに変更を加えることなく，参照される可能性の低いエントリにレジスタ値を書き込むための方策を考える．そして，物理レジスタ番号の割当順に着目した RC の高精度化手法を提案する．

以下，２章では階層型レジスタ・ファイルについて

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

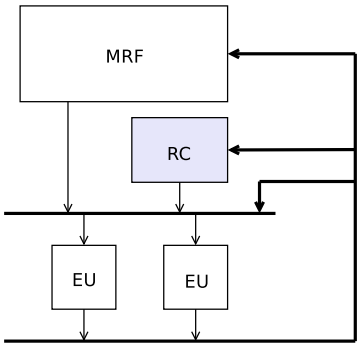

図 1　階層型レジスタ・ファイル

述べる．3 章でダイレクト・マップ方式の RC 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，階層型レジスタ・ファイル，結果バスを持つ．階層型レジスタ・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．したがって，エントリ数は階層化前のレジスタ・ファイルと同一である．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．

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．

そのため，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 種類に分類できる．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．これまでいく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[3),4),9)]．

本論文では，RC のハードウェアの複雑度に大きく関係すると考えられる置き換えポリシーに着目している．RC ミスを削減するためには，できるだけ参照される可能性の低いエントリを選択できる必要がある．また，レジスタの参照には時間的局所性がある[8)]．これらより，**LRU (Least Recently Used) 方式** がもっとも RC ミスを削減できると考えられる．この方式では，レジスタ値を，最も最近使用されていないエントリに書き込むというものである．しかし，RC 書き込み時には最も最近使用されていないエントリを探す必要があり，RC 参照時には，連想検索を行う必要がある．これらは RC のハードウェア複雑度を増加させる．

3. ダイレクト・マップ方式の RC

LRU 方式に対し，RC のハードウェアを最も簡単化できるものとして，ダイレクト・マップ方式がある．この方式は，データキャッシュにおけるダイレクト・マップ方式と同じ考え方を用いており，レジスタ・リネーミング時に割り当てる物理レジスタ番号だけを基に，読み書きする RC エントリを決める．そのため，RC を非常に簡単に実現できる．

ダイレクト・マップ方式の RC では，物理レジスタ番号を RC エントリ数で割った剰余を，RC へのインデクスとする．これは，メモリ階層において，メモリアドレスをキャッシュのエントリ数で割った剰余を，キャッシュへのインデクスとするのと同じ考え方である．例えば，物理レジスタ番号が 9，RC エントリ数が 4 なら，$9 \div 4 = 2...1$ で，剰余は 1 であり，RC へのインデクスは 1 となる．実際の機構としては，エントリ数 n_{rc} の RC は，物理レジスタ番号の下位 i ビットを RC へのインデクスとする．ここで，$i = log_2(n_{rc})$ である．

RC へのインデクスは物理レジスタ番号だけで決まるので，レジスタ・リネーミング時の物理レジスタ番号の割り当て方が重要となる．ある命令がデコードされ，レジスタ・リネーミングが行われるとき，**フリー・リスト**と呼ぶリストから物理レジスタ番号を得て，それを命令のデスティネーション・レジスタに割り当てる．フリー・リストは，現在使用されていない物理レジスタ番号を保持するもので，FIFO で実現される．

プロセッサにおいて，ある物理レジスタが解放される，すなわち，フリー・リストに物理レジスタ番号が戻されるのは，その物理レジスタに対応する論理レジスタが再定義され，再定義された論理レジスタに新たに割り当てられた物理レジスタがコミットされる時である．ここで，プログラムによって，ある論理レジスタは頻繁に再定義され，別のある論理レジスタはあまり再定義されないという状況があるため，物理レジスタの解放のタイミングは，物理レジスタ毎に異なる．これにより，通常フリー・リストの並び方はランダムになる．したがって，物理レジスタ番号はランダムに割り当てられ，RC へのインデクスもランダムとなる．その結果，RC 上で競合が発生し，参照される可能性の高いエントリにレジスタ値を上書きしてしまうと，RC ミスが発生する．こうしてダイレクト・マップ方式の RC では，RC ミスを十分に削減できないという状況が発生する．

これに対し我々は，RC へのインデクスがラウンド

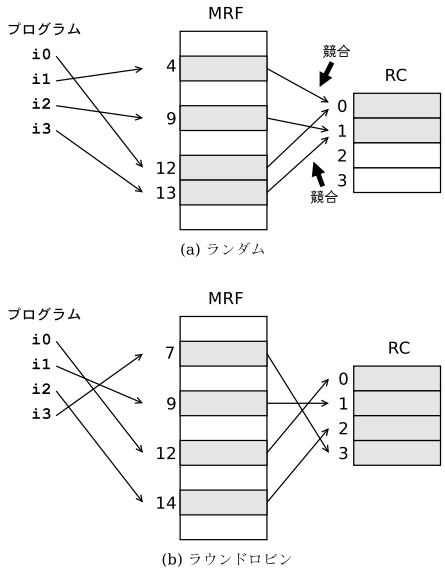

図 2 ダイレクト・マップ方式の RC

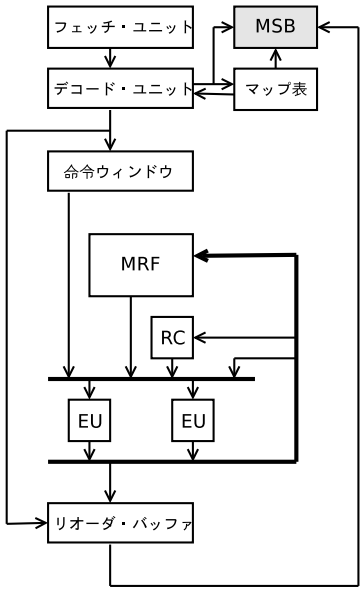

図 3 想定するプロセッサの構成

ロビンに決まるように，物理レジスタ番号を割り当てることを目的とする．これにより，データパスに変更を加えることなく，できるだけ RC 上で競合が発生しないようにすることができる．競合の発生頻度を下げることができれば，参照される可能性の高いエントリにレジスタ値を上書きしてしまうことを避けることができる．

図 2 を用いて，従来のダイレクト・マップ方式の問題点と我々の目的とするアプローチについて説明する．図はダイレクト・マップ方式の RC において，どのようにして RC へのインデクスが決まるかを示す．左側から，プログラム，MRF，RC となる．in は命令を示し，n は命令のフェッチ順を示す．MRF と RC において，エントリの左側の数字は，エントリ番号を示す．MRF は全てのレジスタ値を保持するので，MRF のエントリ番号が，レジスタ・リネーミング時に割り当てる物理レジスタ番号 N_{preg} となる．矢印は命令にどの N_{preg} が割り当てられたかを示す．一方，RC は一部のレジスタ値を保持する．説明を容易にするため，図では，RC エントリ数は 4 とする．レジスタ値は，RC の（N_{preg} % 4）番目のエントリに書き込まれる．矢印はその対応関係を示す．ここで，a % b は，a を b で割った剰余を表す．

ここで，フリー・リストは，6 つの物理レジスタ番号 4, 7, 9, 12〜14 を保持しているとする．そして，このフリー・リストから 4 つの命令 `i1`〜`i4` に対して物理レジスタ番号を割り当てることを考える．図 2 (a) は物理レジスタ番号がランダムに割り当てられる場合，(b) は RC へのインデクスがラウンドロビンに決まるように物理レジスタ番号を割り当てる場合である．

図 2 (a) では，RC へのインデクスがランダムに決まることにより，RC の 0 番目のエントリ，1 番目のエントリで，それぞれ，`i0` と `i1`，`i2` と `i3` が競合している．これにより，参照される可能性の高いエントリにレジスタ値を上書きしてしまう可能性が高くなる．

一方，図 2 (b) から分かるように，我々の目的とする方式では，RC へのインデクスがラウンドロビンに決まるようにしたことにより，競合を解消できている．以上のようにして，データパスに変更を加えることなく，ダイレクト・マップ方式の RC の精度を向上させることを試みる．

4. 提案手法

3 章で述べた目的を実現するためには，レジスタ・リネーミング時に，RC エントリ数で割った剰余が整列された状態になるように，空き物理レジスタ番号を供給しなければならない．以降，RC エントリ数で割った剰余を整列することを，**モジュロ・ソート**と表現する．

一般に，物理レジスタ番号の供給には，フリー・リストが用いられる．したがって，単純に物理レジスタ番号をモジュロ・ソートするには，フリー・リストをサイクルごとにモジュロ・ソートすればよい．しかし，既にバッファ内に保持されている値に対するソート処理は，非常に時間がかかるので現実的ではない．

そこで我々は，解放された物理レジスタ番号を，モジュロ・ソートされた状態で，バッファ内に記録していく機構を提案する．これにより，モジュロ・ソートされた物理レジスタ番号を供給できるようになる．

以下では，まず，提案機構の概要について述べ，次に，その詳細な説明を行う．

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 3** に示す．図は，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

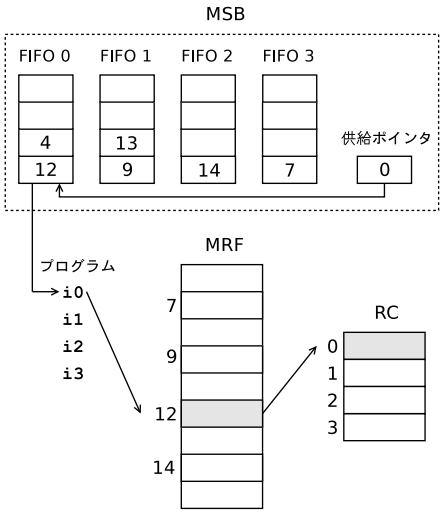

図 4　モジュロ・ソーティング・バッファ(MSB)

ウインドウ，MRF，RC，機能ユニット，リオーダ・バッファを持つ．網掛け部分は提案機構である．提案機構は**MSB（Modulo Sorting Buffer）**と呼ばれるバッファで構成される．MSB は，フリー・リストの替わりに，空き物理レジスタ番号の管理を行う．コミット時にリオーダ・バッファから物理レジスタ番号を受け取り，レジスタ・リネーミング時に物理レジスタ番号を供給する．

4.2　MSB

まず，MSB のハードウェア構成について述べる．次に，MSB のハードウェア量を削減するための方法を示す．最後に，分岐予測ミスに対応するための方法を示す．

4.2.1　ハードウェア構成

MSB は，RC エントリ数と同じ数の FIFO と，どの FIFO から物理レジスタ番号の供給を行えばよいのかを示す**供給ポインタ**で構成される．各 FIFO は，物理レジスタ番号を RC エントリ数で割った剰余が FIFO 番号と等しくなるように，物理レジスタ番号を保持する．一方，供給ポインタは FIFO 番号を保持する．

次に，MSB の更新動作を示す．コミット時に物理レジスタが解放されると，その物理レジスタ番号が MSB に通知される．MSB は，物理レジスタ番号を RC エントリ数で割った剰余に対応する FIFO に，その物理レジスタ番号を書き込む．

最後に，MSB が物理レジスタ番号を供給する動作を示す．レジスタ・リネーミング時に，MSB は供給ポインタの指す FIFO の先頭から，物理レジスタ番号を読み出し，これを命令に供給する．そして，供給ポインタの値をインクリメントする．ただし，供給ポインタが最後の FIFO 番号を保持している場合は，0 にリセットする．

図 4 に，図 2（b）の割り当てを実現する MSB を示す．説明のため，図 2 に示したのと同じプログラム，MRF，RC を示す．図において，MSB は，6 つの物理レジスタ番号 4, 7, 9, 12〜14 を保持している．FIFO 0 の保持する物理レジスタ番号 4 と 12 は，RC エントリ数 4 で割った剰余が，FIFO 番号 0 と等しくなる．他も同様にして，RC エントリ数で割った剰余が FIFO 番号と等しくなる．供給ポインタは 0 を保持している．

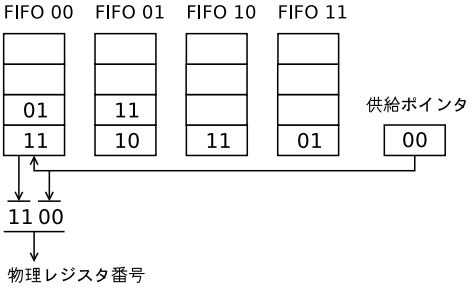

図 5　ハードウェア量を削減した MSB

ここで，i0 からレジスタ・リネーミングを開始するとする．供給ポインタの値が 0 なので，FIFO 0 の先頭から，物理レジスタ番号 12 を供給する．これにより i0 には物理レジスタ番号 12 が割り当てられ，対応する RC エントリは 0 となる．また，供給ポインタの値はインクリメントされて 1 になる．その後は同様にして，i1〜i3 に，それぞれ，物理レジスタ番号 9, 14, 7 が割り当てられていき，対応する RC エントリは，それぞれ，1, 2, 3 となる．以上のようにして，MSB は，モジュロ・ソートされた物理レジスタ番号を供給する．

4.2.2　ハードウェア量の削減

MSB では，物理レジスタ番号を RC エントリ数で割った剰余が，その物理レジスタ番号を保持する FIFO 番号と等しくなる．したがって，RC エントリ数を n_{rc} とすると，物理レジスタ番号の下位 $log_2(n_{rc})$ ビットと FIFO 番号は等しい．

そこで，MSB には，物理レジスタ番号のうち，下位 $log_2(n_{rc})$ ビットを除いたビット列のみを記録することとする．そして，レジスタ・リネーミング時には，記録されたビット列と，供給ポインタの保持する FIFO 番号を連結することで，物理レジスタ番号を復元することとする．以上のようにして，MSB のハードウェア量を削減する．

図 5 に，ハードウェア量を削減した MSB を示す．ビット連結の説明のため，図中の数字は全て 2 進数で表記する．図 5 の MSB は，図 4 の MSB に対応している．RC エントリ数が 4 なので，各 FIFO は，物理レジスタ番号のうち，下位 2 ビットを除いたビット列を保持する．供給ポインタは 00 を保持しているので，レジスタ・リネーミング時には，FIFO 00 の先頭から読み出した 11 と，供給ポインタの 00 をビット連結して，物理レジスタ番号 1100 を読み出す．

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，分岐予測ミスペナルティ 6 サイクル

4.2.3　分岐予測ミスへの対応

MSB では，各 FIFO から順番に物理レジスタ番号を読み出していくことで，モジュロ・ソートされた物理レジスタ番号を供給している．しかし，分岐予測ミスが発生すると，それ以降の命令がフラッシュされるので，その地点において，物理レジスタ番号がモジュロ・ソートされていない状態になる．その結果，RC 上で競合が発生してしまう可能性がある．

この問題を解決するため，分岐予測ミスが判明した時に，供給ポインタの値を，予測ミスした分岐のレジスタ・リネーミング時の値まで巻き戻すこととする．これにより，RC 上の競合の発生を抑制できると考える．

5. 評　　価

5.1　評 価 環 境

SimpleScarar Tool Set[2] のスーパスカラ・プロセッサ用シミュレータに，階層型レジスタ・ファイルと提案機構を組み込んで評価を行った．命令セットには MIPS R10000[7] を拡張した SimpleScarar/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 命令スキップさせた後，100M 命令を実行した．**表 1** に示す測定条件を用いた．

5.2　評価モデル

以下に示すモデルに対して評価を行った．

- **DM** モデル: ダイレクト・マップ方式の RC を用いるモデルである．
- **MSB** モデル（提案手法）: ダイレクト・マップ方式の RC を用い，MSB によって物理レジスタ番号のモジュロ・ソートを行うモデルである．
- **MSB-r** モデル（提案手法）: ダイレクト・マップ方式の RC を用い，MSB によって物理レジスタ番号のモジュロ・ソートを行い，分岐予測ミス時に供給ポインタの巻き戻しを行うモデルである．

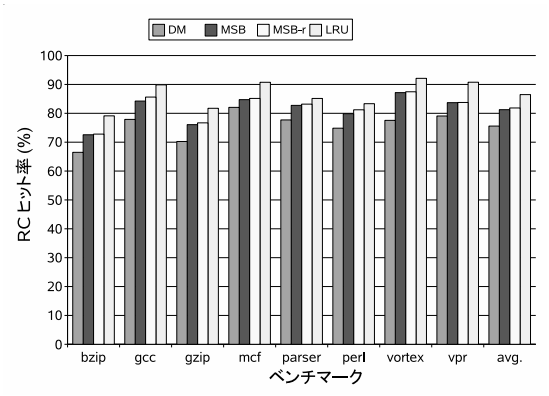

図 6　RC ヒット率（エントリ数 32）

- **LRU** モデル: LRU 方式の RC を用いるモデルである．

どのモデルにおいても，書き込みポリシーとして，命令の生成した値を全て RC に書き込む All Cache 方式を用いる．

5.3　測 定 結 果

図 6 に，RC エントリ数を，論理レジスタ数と同じ 32 としたときの各モデルの RC ヒット率を示す．図の縦軸は RC ヒット率を示し，横軸はベンチマークおよびベンチマーク平均（avg.）を示す．4 本で組になっている棒グラフは，左から順に DM モデル，MSB モデル，MSB-r モデル，LRU モデルである．

図 6 より，LRU モデルは，DM モデルに対し，ヒット率が平均 10.9%ポイント高い．しかし，このモデルでは，RC 書き込み時には最も最近書き込まれていないエントリを探す必要があり，RC 参照時には連想検索を行う必要がある．

また，MSB モデルは DM モデルに比べて，最大 9.6%ポイント，ベンチマーク平均で 5.7%ポイントヒット率が向上していることが分かる．一方，MSB-r モデルは，MSB モデルよりヒット率が高いが，その差はわずか 1%ポイント未満である．これらより，物理レジスタ番号をモジュロ・ソートすることによって，ヒット率が向上することが分かる．データパスに変更を加えていないので，アクセス時間への悪影響はないといえる．しかし，供給ポインタの巻き戻しによる効果は低い．これは，分岐予測ミスによってフラッシュされる命令数が多く，RC のほとんどのエントリが無効になってしまうためと考えられる．

次に，RC エントリ数を 8, 16, 32, 64 と変化させたときの RC ヒット率を**図 7** に示す．図の縦軸は RC ヒット率のベンチマーク平均を示し，横軸は RC エントリ数を示す．4 本で組になっている棒グラフは，左から順に DM モデル，MSB モデル，MSB-r モデル，LRU モデルである．

図 7 より，MSB モデルの RC ヒット率は，RC エントリ数が増えるにつれて，DM モデルに対し，2.0%, 4.2%, 5.7%, 4.1%ポイントと増加していく．一方，MSB-r モデルでは，2.1%，4.5%，6.3%，5.0%ポイン

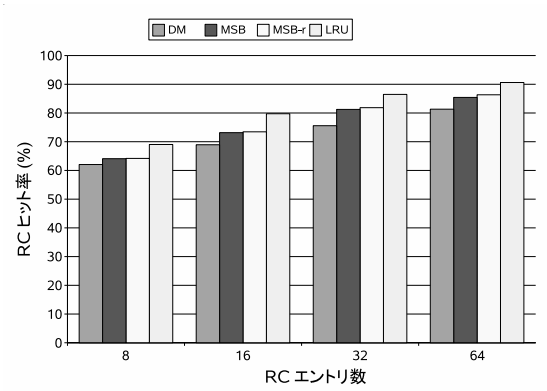

図 7 RC エントリ数と RC ヒット率の関係

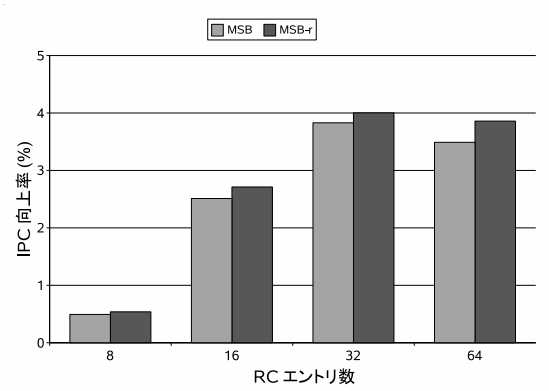

図 8 エントリ数と性能向上率の関係

トと増加していく．これらより，MSB モデルと MSB-r モデルの差は，RC エントリ数が増えるほど開くことがわかるが，その差は 1%未満であり，大きいとはいえない．また，MSB モデル，MSB-r モデル共に，DM モデルに対するヒット率の向上が最大となるのは，RC エントリ数が 32 の場合であることが分かる．この理由は，RC エントリ数が 32 を超えると，レジスタ・リネーミング時に物理レジスタ番号がランダムに供給されても，RC 上での競合が発生し難くなることにあると考えられる．

5.4 性能に与える効果

本節では，提案手法による RC ヒット率の向上が，IPC（プロセッサ性能）に与える効果について評価を行う．

図 8 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 DM モデルに対する性能向上率をベンチマーク平均で示し，横軸は RC エントリ数を示す．2 本で組になっている棒グラフは，左から順に MSB モデル，MSB-r モデルである．

図 8 から，MSB モデルと MSB-r モデルはほぼ同一の性能向上率であることが分かる．提案方式による性能向上率は RC エントリ数が 32 のとき最大となり，MSB モデルで 3.8%，MSB-r モデルで 4.0%の性能向上を達成している．

以上のように，RC ヒット率の向上は，プロセッサ性能の向上に貢献していることが分かった．

6. ま と め

本論文では，階層型レジスタ・ファイルを導入したプロセッサにおいて，レジスタ・リネーミング時の物理レジスタ割当順に着目することで，ダイレクト・マップ方式の RC のデータパスを変更することなく，RC ヒット率を向上させる手法を提案した．提案手法では，モジュロ・ソート・バッファと呼ぶ機構を導入し，RC へのインデクスがラウンドロビンに決まるように物理レジスタ番号の割り当てを行う．

評価の結果，RC エントリ数が 32 のとき，ダイレクト・マップ方式の RC に対して，RC ヒット率を最大 9.9%ポイント，平均 6.3%ポイント向上できることが分かった．また，RC ヒット率の向上により，最大 6.1%，平均 4.0%性能が向上することが分かった．

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] R. Balasubramonian, et al., “Reduing the Complexity of the Register File in Dynamic Sperscalar Processors,” *MICRO-34*, pp.237-248, Dec. 2001.

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

[3] J. A. Butts, et al., “Characterizing and Predicting Value Degree of Use,” *MICRO-35*, pp.15-26, Nov. 2002.

[4] J. A. Butts, et al., “Use-Based Register Caching with Decoupled ndexing,” *ISCA-31*, pp.302-313, Mar. 2004.

[5] J. L. Cruz, et al., “Multipule-Banked Register File Architectures,” *ISCA*, pp.316-325, Jun. 2000.

[6] R. Gonzalez, et al., “A Content Aware Integer Register File Organization,” *ISCA-31*, pp.314-324, Mar. 2004.

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

[8] S. Vajapeyam, et al., “Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences,” *ISCA-24*, pp.1-12, Jun. 1997.

[9] R. Yung, et al., “Caching Processor General Registers,” *ICCD’95*, pp.307-312, October 1995.