一致経路長の短縮による Renamed Trace Cache のエネルギー効率向上

塩谷亮太[†]　安藤秀樹[†]

Out-of-order スーパスカラ・プロセッサでは命令間の偽の依存を取りのぞくためにレジスタ・リネーミングを行う．このためのリネーム・ロジックは out-of-order スーパスカラ・プロセッサの中でも最も高コストな部分の 1 つであり，非常に大きな電力を消費する．これを解決するため，我々はこれまでにリネーミング済みの命令列をキャッシュする Renamed Trace Cache（RTC）を提案してきた．従来の RTC では，リネーミングに関わる消費エネルギーを大きく減らせる一方，キャッシュのハードウェア量増加によるオーバーヘッドや容量効率低下による性能低下の問題があった．これに対し，本論文ではリネーミング結果のキャッシュをプロデューサまでの距離が近いオペランドのみに制限する手法を提案する．距離を制限することは生成トレース数の大幅な削減——すなわちヒット率の大幅な向上につながり，上記の問題を解決することができる．評価では，通常のプロセッサと比較して性能低下を 0.3%にとどめながら，リネーミングに関わる消費エネルギーを 53%削減できることを確かめた．

Energy Efficiency Improvement of Renamed Trace Cache through Reduction of Matching Path Length

RYOTA SHIOYA[†] and HIDEKI ANDO [†]

Out-of-order superscalar processors rename registers to resolve dependencies between instructions. A renaming logic is one of the highest cost components of out-of-order superscalar processors and it consumes large amounts of energy. We have proposed Renamed Trace Cache(RTC) that caches and reuses renamed instructions to reduce energy consumption for renaming. The conventional RTC reduces energy consumption for register renaming, but it does not effectively reduce total energy consumption in a processor and it may degrade performance. In this paper, we propose a method that caches only renamed operands near their producers in RTC and it improves cache efficiency and reduces hardware overhead. Our evaluation results shows that the performance degradation of our proposal is only 0.3% and it reduces the energy consumption by 53% compared to a conventional processor.

1. はじめに

Out-of-order スーパスカラ・プロセッサでは命令間の偽の依存を取り除くために**レジスタ・リネーミング**を行う．このレジスタ・リネーミングは**レジスタ・マップ・テーブル（Register Map Table: RMT）**と呼ぶ表を読み書きすることにより行われる．この RMT は非常に大きな回路であり，2.1 節で詳しく述べるように，その消費エネルギーが深刻な問題となっている．

この RMT の問題を解決するため，我々はこれまでに **Renamed Trace Cache（RTC）**を提案した[1]．RTC は，リネーミング済みの命令列をキャッシュする Trace Cache（TC）[2] である．リネーミングを行うのは RTC にミスした時のみであるため，RMT の処理幅，すなわちポート数を少なくすることができる．RAM の回路面積はポート数の 2 乗に比例して大きくなるため[3]，このポート数の削減は RMT を大幅に縮小させる．

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

Renamed Trace Cache の問題

RMT が縮小される一方，RTC 方式では容量効率が大きく低下してしまう．同様の問題は通常の TC 方式でも発生するが，RTC 方式の場合ではより深刻である．通常の TC 方式では，ある基本ブロックを先頭とするトレースは複数あるため，その容量効率は低下する．これに対し RTC 方式では上記の場合に加え，1 つのトレースに対しその後方のトレース外の経路にも対応して別途リネーミングを行った命令列をトレースとしてキャッシュする必要がある．トレースが依存する経路長が伸びると，生成されるトレース数は指数関数的に増えるため，このトレース外の経路による影響は非常に大きく，容量効率を大きく低下させてしまう．5 節の評価で述べるように，これにより RTC 方式では通常の命令キャッシュとくらべて最大 16%も性能を低下させてしまっている．

また，RTC 方式では経路の一致を確かめるために，タグ・アレイに間接分岐のターゲット・アドレスを格納する必要がある．これによる回路資源の増加は非常に大きく，5 節で述べるように，RMT 縮小による消費

エネルギー削減の効果を相殺してしまっている．

提 案 手 法

本論文では，RTC に格納するオペランドのリネーミング結果を，そのトレース外プロデューサまでの距離が近いものに制限する手法を提案する．それ以外のオペランドについては論理レジスタ番号をキャッシュし，フェッチ後に RMT を用いてリネームする．生成されるトレース数はトレース外プロデューサまでの距離に対して指数関数的に増大するため，距離を制限することは生成トレース数の大幅な削減—— すなわち容量効率の大幅な向上につながる．

リネームされずに論理レジスタ番号として格納されたオペランドは RMT を用いてリネーミングが行われるが，これは大きな問題とはならない．一般に，大多数のソース・オペランドのプロデューサは数命令程度しか離れておらず，このため論理レジスタ番号として格納されるオペランドの数は十分少ない．したがって，RTC 方式がミス時のためにもとから備えている少ポートの RMT によって十分リネーミング可能であり，それによる消費エネルギーも小さい．

提案手法では前述した間接分岐のターゲット・アドレス格納による問題も解決する．提案手法では依存するトレース外の経路を短く制限するため，この経路内に間接分岐が現れる頻度は十分低くなり，ターゲット・アドレスの格納を省略しても大きな性能低下にはつながらない．これにより，ハードウェア量を削減し，消費エネルギーを大きく下げることができる．

5 節の評価では，従来の命令キャッシュを持つプロセッサと比較して，性能低下を 0.3%にとどめながら，リネーミングに関わる消費エネルギーを 53%削減できることを確かめた．また，従来の RTC と比較して，性能を 3.6%向上させながら，消費エネルギーを 64%削減できることを確かめた．

本論文の以降の構成は以下のとおりである．2 節では，提案手法の背景として RTC について説明し，3 節では，その問題点についてまとめる．4 節では提案手法について説明し，5 節ではシミュレーションにより定量的な評価を行う．6 節では関連研究について説明し，7 節でまとめる．

2. Renamed Trace Cache

以下では，まず背景として RMT の消費エネルギーについて説明し，その後 RTC の方式を説明する．

2.1 RMT の消費エネルギー

レジスタ・リネーミングは，RMT を読み書きすることにより行われる．RMT は，論理レジスタ番号から物理レジスタ番号への対応を保持する表であり，一般に多ポートの RAM や CAM によって構成される．[4),5)]

この RMT は非常に大きな回路である．一般的な 3 オペランド形式の命令セットの場合，RMT には 1 命令

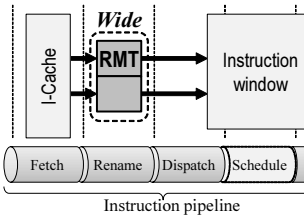

図 1 通常のプロセッサのフロントエンド

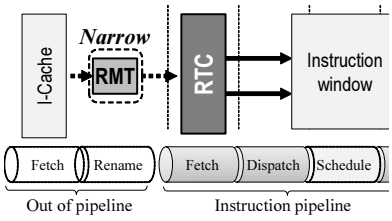

図 2 RTC を持つプロセッサのフロントエンド

あたり 4 ポートが必要となる[6)]．このため，4 命令同時発行可能なプロセッサでは，RMT を構成する RAM のポート数は 16 にもなる．多ポートの RAM の回路面積はポート数の 2 乗に比例するため[3)]，RMT の回路面積は，その容量に対し非常に大きなものとなる．実際，Alpha 21464 プロセッサでは，リネーム・ロジックの回路面積は L1 データ・キャッシュ（64 KB）よりもはるかに大きい[5)]．

巨大な RMT による消費電力と，それによる発熱は深刻である．RAM の消費電力は，一般にその回路面積とアクセスの頻度に比例して大きくなるためである[3)]．ほぼ全ての命令は RMT に対して複数回アクセスするため，そのアクセス数は非常に多い．RMT と同様にアクセス頻度の高いレジスタ・ファイル（Register File: RF）と比較しても，RMT はその 2.3 倍ものアクセスを受ける[1)]．このため RF と比較して，RMT は面積あたりにおいてより大きな電力を消費する．

2.2 RTC 方式の概要

RTC 方式はリネーミング済みのトレースをキャッシュして再利用することにより，上記の RMT による問題を解決する．**図 1** と**図 2** に，それぞれ通常のスーパスカラ・プロセッサと RTC を備えるプロセッサのフロントエンドを示す．通常のプロセッサでは，命令をフェッチし，RMT を用いてリネーミングを行い，その後命令ウィンドウにディスパッチする．これに対し，RTC 方式では，RTC からフェッチした命令はリネーミング済みであるため，そのままディスパッチされる．リネーミングは RTC ミスに伴うトレース生成時にのみ行われるため，性能を下げることなく RMT のポート数を削減できる．この RMT のポート数の削減は，回路面積と消費エネルギーの大幅に削減につながる．

2.3 RTC の命令の方式

一般に，リネーミング結果を再利用することは難し

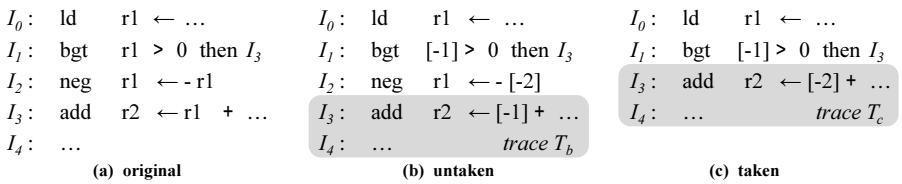

図 4 dualflow 形式への変換の例

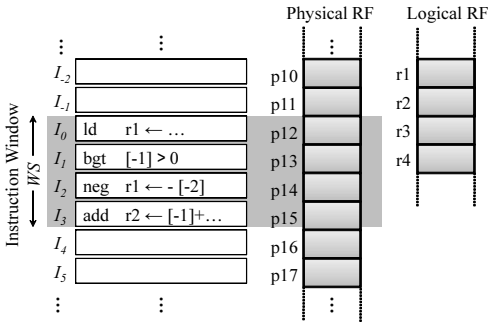

図 3 RTC 方式のレジスタ・ファイルの構成

い．物理レジスタはフリー・リストから割り当てられるため，命令に割り当てられる物理レジスタの番号は毎回異なる．また，ソース・オペランドのプロデューサは，その命令までの実行経路によって変化するため，参照先の物理レジスタも経路ごとに変化する．これらのため，通常のレジスタ・リネーミングではその結果が毎回異なり，それを再利用することができない．

RTC 方式では物理レジスタの割り当てと参照を工夫することにより，リネーミングの結果を再利用可能にしている．RTC 方式では，物理レジスタは物理 RF 上でプログラム・オーダに従ってシーケンシャルに割り当てられる．ソース・オペランドは，n 命令前の実行結果と言う形で命令間——すなわちこの物理 RF 上での**変位**で指定する．プロデューサへの命令間の相対的な変位は，実行経路が同じであった場合は常に一定である．この変位で表されたオペランドを経路ごとにキャッシュすることにより，リネーミングの結果を再利用する．以下ではこの変換形式を **dualflow 形式**，また変換された命令を **dualflow 命令**と呼ぶ．この dualflow 形式への変換は，通常のレジスタ・リネーミングと同様の RMT を用いて行うことができる[1]．

2.4 RTC のレジスタの方式

RTC では，物理と論理の 2 種類の RF を用いる．**図 3** に，RTC の RF の構成を示す．同図で I_{-2} から I_5 は連続する命令を，p10 から p17 は物理レジスタ番号を，r1 から r4 は論理レジスタ番号をそれぞれ表す．

物理 RF はリング状の構造を取り，各エントリは命令に対してシーケンシャルに割り当てられる．たとえば図 3 では，プログラム・オーダに従って並んだ I_{-2} から I_5 までの命令に対し，それぞれの右側にある p10 から p17 までのエントリが割り当てられている．前述したように，dualflow 形式のソース・オペランドは命令間の変位として表現されており，自身の物理レジスタ番号にこの変位を加算することによって，ソース・オペランドの物理レジスタ番号を得ることができる．

論理 RF は，命令ウィンドウからリタイアした命令の実行結果を保持する．命令は，リタイア時にその実行結果を論理 RF に対してインオーダに書き込む．プロデューサが命令ウィンドウのサイズ WS ☆ 以上離れており，自身が命令ウィンドウに入る際にプロデューサがリタイアしていることが保証される場合には，論理 RF から値を読み出す．

2.5 dualflow 命令のキャッシュ

2.2 節で述べたように，dualflow 命令では命令間の変位を用いてソース・オペランドを指定する．この変位は，依存関係にある命令間の経路が異なる場合には変化してしまう．このような場合の例を**図 4** に示す．同図の (a) は，r1 にロードした値の絶対値をとり，その結果を加算するコードである．同図の (b) と (c) は，これを dualflow 形式に変換したものであり，それぞれ I_1 の分岐が untaken であった場合と taken であった場合の変換結果である．I_1 が untaken であった場合，I_3 は 1 命令前の I_2 に依存するため，図 4（b）のように，ソース・オペランドの変位は $[-1]$ となる．一方，I_1 が taken であった場合，I_3 は 2 命令前に実行された I_0 に依存するため，図 4（c）のように，ソース・オペランドの変位は $[-2]$ となる．

RTC 方式では，この dualflow 命令を経路ごとに生成し，それぞれ区別してキャッシュする．ここでトレースとは，実行時の動的な順に並べられた命令列のことを指す．このキャッシュには TC と同様の仕組みを用いる．TC 方式と異なるのは，TC 方式が**トレース内**の経路ごとにトレースを区別するのに対し，RTC 方式ではそれに加えて**トレース外**の後方の経路にも応じてトレースを区別する点である．たとえば図 4 の I_3 と I_4 からなるトレースをキャッシュする場合では，直前までに（b）の経路（I_0，I_1，I_2）を実行したか，あるいは（c）の経路（I_0，I_1）を実行したかに応じて，それぞれ異なるトレース T_b と T_c を生成し，区別してキャッシュする．

RTC 方式では，このトレースの識別に TC 方式と同様の経路情報を用いる．経路情報とは，具体的には条件分岐命令の分岐方向と，間接分岐のターゲット・ア

☆ WS はリオーダ・バッファのサイズである．

ドレスである．RTC 方式では，これらの経路情報と，経路の先頭命令アドレスをタグ・アレイに格納する．

2.6 RTC からのフェッチ

フェッチ時はこの経路情報を PC と分岐の履歴から生成し，タグ・アレイに格納された経路情報と比較して RTC のヒット・ミス判定を行う．ヒット時は得られたトレースをそのままディスパッチする．ミス時はフロントエンドをストールさせ，通常の命令キャッシュから命令をフェッチし，RMT を用いて dualflow 命令に変換してからディスパッチする．また，変換された dualflow 命令はその後コミット時にトレースにまとめられフィルされる．この時使用する RMT はそのポート数を制限しているためスループットが低下するが，RTC のヒット率が十分高ければ問題とはならない．RTC を構成するメモリの回路面積はポート数の 2 乗に比例して大きくなるため[3]，このポート数の削減は RMT を大幅に縮小させる．

3. Renamed Trace Cache の問題

既存の RTC の主な問題は，(1) キャッシュの容量効率低下と，(2) タグ部分のハードウェアの大幅な増加である．本節では，これらについて順に説明する．

3.1 容量効率の低下

RTC 方式では容量効率の低下が問題となる．TC 方式と比較して，RTC 方式では 1 つのトレースに対しその後方のトレース外の経路にも対応して別途リネーミングを行った命令列をキャッシュする．トレースが依存する経路長が伸びると生成されるトレース数は指数関数的に増えるため，容量効率を大きく低下させる．

このことを，**図 5** を用いて説明する．同図は制御フロー・グラフであり，ノードは命令を表す．ノードのうち，斜線によって塗りつぶされたものは分岐命令である．以下では，I_{t_start} から I_{t_end} に至るトレースのフェッチについて，TC と RTC の場合を比較しながら説明する．

(1) TC 方式の場合： I_{t_start} から I_{t_end} に至るトレース内の経路がタグ・アレイ中の経路情報と一致した場合にヒットする．この時の経路情報の長さはトレース長である 3 命令となる．PC は I_{t_start} を指しているので，それ以降の経路については分岐予測の結果を用いる．

(2) RTC 方式の場合：上記トレース内の経路に加え，トレース外の後方の経路も含めてタグ・アレイ中の経路情報と一致した場合にヒットする．この時のトレース外の経路情報の長さは，トレース内にあるソース・オペランドの中で最も長い変位の長さとなる（以降，これを**一致経路長**と呼ぶ）．たとえば I_{t_start}，I_{t_br}，I_{t_end} のソース・オペランドが I_{p1} の実行結果を参照していた場合，経路情報にはトレース内の 3 命令分に加え，I_{p1} から I_{t_start} に至るまでの後方の 3 命令分が追加で必要となる．

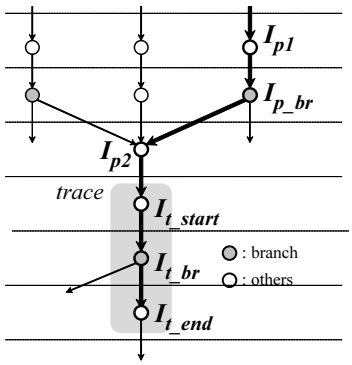

図 5 制御フローと経路情報の長さ

このトレース外の経路情報の長さは最大で命令ウィンドウのサイズ WS に制限される．これは 2.4 節で述べたように，WS 以上離れた命令は必ずリタイアしており，結果が論理 RF にあることが保証されているためである．

このような違いにより，両者では生成されるトレース数が大きく異なる．たとえばトレース長が 3 命令である通常の TC 方式では，トレース内の 3 命令全てが条件分岐であるような最悪の場合でも，同一の先頭アドレスに対して生成されるうるトレースの数は $2^3 = 8$ 通りである．一方 RTC 方式では，上記に加えてトレース外の後方の経路にも応じてトレースが生成される．この時の後方の経路長は最大で WS となるが，近年のプロセッサでは，WS は 100 命令以上にも及ぶため[7]，最悪の場合は 2^{100} 以上のトレースが生成される．このように生成されるトレース数が大幅に増えてしまうため，RTC 方式では容量効率が大きく低下する．

3.2 タグのオーバーヘッド

もう 1 つの問題は，タグへの経路情報格納によるハードウェア量の増加である．この問題は，TC 方式と比べて格納する分岐方向数が大きく増えることと，間接分岐ターゲット・アドレスを格納することによる．

(1) 分岐方向情報の増加： RTC 方式の経路情報は，通常の TC 方式で用いられるトレース内の分岐方向情報と同様のものである．前述したように，RTC 方式では通常の TC 方式と比較して一致経路長がより長くなるため，そのためにより大きな容量が必要となる．

(2) 間接分岐ターゲット・アドレスの格納：

TC 方式では通常，間接分岐のターゲット・アドレスはタグに格納されない．これは，このターゲット・アドレスが数十ビット程度と非常に大きいためである．このため，通常の TC 方式では経路情報にターゲット・アドレスは用いず，間接分岐が現れた場合はそこでトレースを終了する．間接分岐の出現頻度は通常十分低いため，そこでトレースを終了したとしても大きな性能低下には繋がらない．

一方 RTC 方式の場合，タグに間接分岐ターゲット・アドレスを格納しない場合は性能が大きく下がる[1]．これは RTC 方式では，ヒット・ミス判定がトレース外の後方の経路に依存しているためである．タグにターゲット・アドレスを格納しない場合，一致経路長以内の後方に間接分岐があるトレースは経路の一致を保証できないため，RTC に格納できない．この結果，間接分岐が現れると，その前方にある一致経路長分の命令が RTC に格納不能となる．このため，RTC 方式では数個程度のターゲット・アドレスをタグに格納する．それを超えた数の間接分岐が現れた場合は，キャッシュにトレースは格納されず，RMT を用いてリネーミングが行われる．これまでの研究では，タグには 2 個程度のターゲット・アドレスを格納することで，性能低下を十分抑えられることがわかっている．

3.3 追加ハードウェア量の検討

上記で述べた追加ハードウェアの量は非常に大きい．以下ではこのことを具体例により説明する ☆．

(1) データ（トレース）： 論理レジスタ番号と変位による表現をソース・オペランドでは排他的に使用するため，それらを格納するフィールドは共用できる．変位の最大値を 127（7 ビット）とした場合，論理レジスタ数を 32（5 ビット）とすると 2 ビットの容量が追加で必要となる．命令長を $32+2\times2=36$ ビット，トレースの長さを 4 命令とした場合，1 つのトレースには $4\times36b/8=18$ バイトの容量が必要となる．

(2) タグ： タグ部分には，経路の先頭の命令アドレス，間接分岐のターゲット・アドレス，条件分岐の分岐方向が格納される．命令アドレスを 48 ビット（6 バイト）☆☆，格納するターゲット・アドレスを 2 つ，命令ウィンドウ・サイズを 128 とすると，タグ部分には合計で $6B+6B\times2+128b/8=34$ バイトもの容量が必要となる．このように，タグの容量はトレースのデータと比べて倍程度の大きさであり，通常のキャッシュや TC の場合と比べて非常に大きい．

4. 提 案 手 法

我々は，RTC に格納するオペランドのリネーミング結果を，そのトレース外プロデューサまでの距離が近いものに制限する手法を提案する．以下では，提案手法のモチベーションとして，オペランドごとのプロデューサへの距離（以下ではこれを**参照距離**と呼ぶ）について述べる．その後，提案手法の詳細と効果について説明する．

4.1 参照距離の分布

通常，多くのソース・オペランドは比較的近くにある命令の結果を参照している．**図 6** は，SPECCPU INT 2006[8] におけるソース・オペランドの参照距離の分布

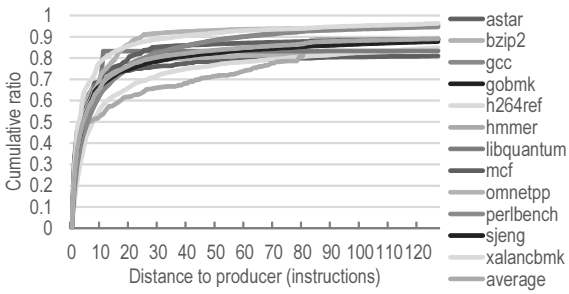

図 6 オペランドごとのプロデューサへの距離の分布

である ☆☆☆．横軸はソース・オペランドごとの参照距離（命令数），縦軸は全ソース・オペランドに対する累積の割合を表す．たとえば，横軸 20 かつ縦軸 0.7 の点は，参照距離が 20 命令以下であるソース・オペランドが全体の 7 割を占めることを表す．グラフより，ソース・オペランドの大部分が，10 ～ 20 命令以内にあるプロデューサを参照していることがわかる．

これに対し，トレースの一致経路長（3.1 節）は個々の参照距離よりも大幅に長くなってしまう．既存の RTC 方式では，一致経路長はトレース内の最も参照距離が長いオペランドの変位となるためである．このため，大部分のオペランドの参照距離が 10 ～ 20 命令以内であるにも関わらず，既存の RTC 方式では一致経路長の長さは平均で 80 命令以上にもなる．

4.2 参照距離によるキャッシュの制限

我々は，RTC に格納するオペランドのリネーミング結果を，そのトレース外プロデューサまでの距離が近いものに制限する手法を提案する．それ以外のオペランドについては論理レジスタ番号を格納し，フェッチ後に RMT を用いてリネームする．

以下では，既存の RTC 方式と比較を行いながら，提案手法の動作を説明する．**図 7** と**図 8** に，既存の RTC 方式と提案する RTC 方式を持つプロセッサのフロントエンドを示す．既存手法では，RTC にヒットした場合は得られたトレースをそのままディスパッチする．ミスした場合は命令キャッシュから命令を読み出し，リネーミングを行う．命令キャッシュや RMT へアクセスするステージは通常の命令パイプラインの外に置かれ，これらはミス時にのみ動作する．一方提案手法では RTC のヒット/ミスに関わらず，命令キャッシュと RMT へのアクセスを行うステージを用意し，以下のように動作する：

(1) RTC にヒットした場合： 続く Fetch-I ステージ（図 8）では命令キャッシュにはアクセスせず，NOP としてそのまま通過する．続く Rename ステージでは，トレース内に論理レジスタ番号として格納されているオペランドのみ，リネーミングを行う．

(2) RTC にミスした場合： 続く Fetch-I ステージで命令キャッシュにアクセスする．その後，続く Rename

☆ これらは厳密ではなく，ある程度簡単化している．
☆☆ 物理アドレスを格納することを想定している
☆☆☆ 評価条件については， 後の 5 節で述べるものと同じである．

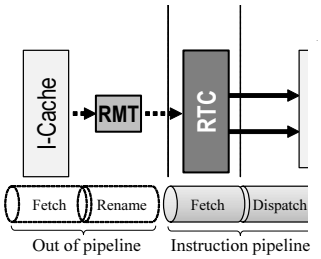

図 7 既存の RTC 方式のフロントエンド

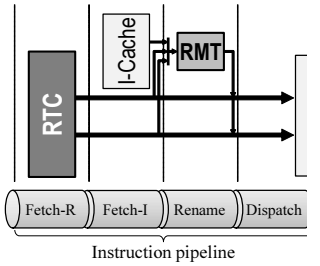

図 8 提案する RTC 方式のフロントエンド

ステージで リネーミング を行う．

提案手法の RMT は，既存の RTC 方式のそれと同様にポート数を減少させる．トレースを生成して RTC に格納する際は，このポート数を超えたリネーミングの必要がないようトレースの長さを制限する．すなわち，RMT のポート数を超えたリネーミングが必要になる場合，そこで一旦トレースを切り，残りの命令は次のトレースに格納するようにする．

4.3 提案手法の効果

提案手法は，従来の RTC 方式による RTC の容量効率低下とハードウェア量増加の双方を解決する．

4.3.1 容量効率の改善

提案手法では，トレースの一致経路長を短距離に限定するため，キャッシュの容量効率が向上する．このことを，**図 9** の例を使って説明する．同図は図 5 と同様の制御フローである．ここで，図内の各命令のソース・オペランドは 1 つであるものとする．今，I_{t0} と I_{t1} の 2 命令からなるトレースに着目する．このトレースに至る経路は 3 本あるが，I_{t0} のプロデューサは，どの経路を通った場合でも I_{p3} の 1 つである．一方，I_{t1} のプロデューサは経路ごとに異なり，それぞれ I_{p0}，I_{p1}，I_{p2} となる．

既存の RTC 方式では，一致経路長はトレース内で最も参照距離が長いオペランドへの変位となる．このため，I_{t0} に至る 3 つの経路ごとにそれぞれ異なるトレースが生成される．

これに対し，提案手法で一致経路長を 1 命令に限定した場合を考える．I_{t0} のプロデューサはトレースの後方 1 命令以内にあるため，リネーミング結果がキャッシュされる．一方，I_{t1} のプロデューサはどれも 1 命令より離れているため，リネーミング結果はキャッシュされず，論理レジスタ番号がキャッシュされる．ここで，I_{t0} に至る 1 命令前の経路は I_{p3} の 1 つであるため，生成される可能性があるトレースは 1 通りである．このようにして生成されるトレースの数が削減されるため，キャッシュの容量効率が向上する．

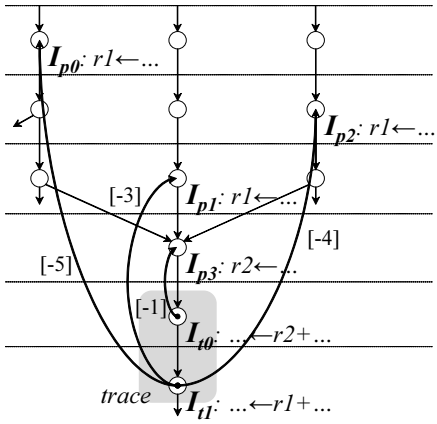

図 9 命令ごとの参照距離

3 節で述べたように，トレースの一致経路長に応じて，生成されるトレース数は指数関数的に増加し，RTC の容量効率を低下させる．5 節で述べるように，提案手法では一致経路長を 6 命令程度にまで短縮できるため，生成されるトレースの数は劇的に削減される．

論理レジスタ番号として格納されたオペランドは RMT によりリネーミングが行われるが，大きな問題とはならない．前述したように，大多数のソース・オペランドのプロデューサは数命令程度しか離れておらず，このため論理レジスタ番号として格納されるオペランドは十分少ない．したがって，RTC 方式がミス時のために元から備えている少ポートの RMT によって十分リネーミング可能であり，それによる消費エネルギーも小さい．

4.3.2 タグのハードウェア量削減

提案手法では大きな性能低下を伴わずに，タグへの間接分岐のターゲット・アドレス格納（3.2 節）を省略できる．3.2 節で述べたように，既存の RTC 方式において大きな性能低下を起さないためには，タグに 2 個程度の間接分岐ターゲット・アドレスを格納する必要がある[1)]．一方，提案手法では一致経路長を 6 命令程度の少数にすることができる．このため，タグにターゲット・アドレスを格納せず，依存する経路上に間接分岐があらわれた場合はキャッシュしない場合でも，その影響はたかだか 6 命令程度である．3.2 節で述べたように，タグへのターゲット・アドレス格納は大きなオーバーヘッドとなるため，これの省略は回路資源を大きく削減する．また，これにより，消費エネルギーや遅延も大きく削減される．

5. 評　　価

シミュレーションにより，提案手法を評価した．以下では評価環境について述べた後，性能や消費エネルギーなどの評価結果を説明する．

5.1 評 価 環 境

性能の評価は，プロセッサ・シミュレータ**鬼斬弐**[9]を用いて行った．評価にはSPECCPU 2006[8]に含まれる全ベンチマークを用いた．ベンチマークのコンパイルにはgcc 4.2.2を用い，コンパイル・オプションには“−O3”を指定した．入力データ・セットにはrefを用い，プログラムの先頭1 G命令をスキップして，続く100 M命令の評価を行った．

消費エネルギーの評価はCACTI 6.5[10]を用い，32 nmテクノロジを想定して行った．

5.2 評価モデル

表 1にベースラインとなる構成を示す．この構成は，IBM Power 7[7]と同様にフロントエンドの幅を6命令，発行幅を8命令とし，その他のパラメータについても近い構成となるよう選んでいる．このベースラインに対し，以下のモデルを実装して評価した．

(1) Base: 通常の命令キャッシュを備えたモデル．
(2) TC: TCを備えたモデル．
(3) RTCConv: 既存のRTCを備えたモデル．
(4) RTCProp: 提案手法によるRTCを備えたモデル．

TCとRTCの構成については**表 2**に示すとおりである．TCやRTCのモデルでは，トレース中に含むことのできる分岐の数は2とした．また，TCやRTCのヒット/ミスは1サイクル目でわかるものとし，命令キャッシュへのアクセスはその後行うものとした．RTCConvにおいて，タグに格納する間接分岐のターゲット・アドレスは2つとした．

表 1 ベースラインの構成

Name	Parameter
ISA	Alpha
fetch width	6 inst.
rename width	6 inst.
issue width	8 inst.
issue queue	64 entries, unified
execution unit	int:4, fp:4, mem:2
ROB	128 entries
branch predictor	8 KB g-share, 2K entries BTB
L1C (D/I)	32 KB (34.3 KB☆), 8 way, 64 B/line, 2 cycles
L2C	256 KB, 8 way, 64 B/line, 8 cycles
L3C	4 MB, 8 way, 64 B/line, 24 cycles
main memory	200 cycles

表 2 TC と RTC の構成

Name	Parameter
TC	1 K traces (29 KB☆), 8 way, trace:6 inst., 3 cycles
RTCConv	1 K traces (51.4 KB☆), 8 way, trace:6 inst., 3 cycles
RTCProp	1 K traces (30.9 KB☆), 8 way, trace:6 inst., 3 cycles

5.3 提案手法の構成の検討

提案手法には多数のパラメータがあり，それらが相互に影響する．探索空間を絞るため，本節ではパラメータを変化させて性能を評価し，以降で使用するパラメータを決定する．

まず，RTCの容量を0.5 Kトレースから4 Kトレースまで変化させて性能を評価した．**図 10**はRTCPropのBaseに対する相対IPCである．グラフ内の各折れ線は全ベンチマークの結果の幾何平均と最低値にそれぞれ対応する．同図の評価では，RTCの一致経路長（3.1節，4.2節）は6命令に固定し，RMTの処理幅は2命令（4オペランド）としている．このグラフより，RTCに1 Kトレースの容量があればBaseに対して平均的に性能を低下させず，最悪の場合でも性能低下を10%以内におさめられることがわかる．したがって，以降ではこの容量を基準として評価を行う．

次に，**図 11**に一致経路長とRMTの処理幅を変化させた場合のBaseに対する相対IPCを示す．同図は全ベンチマークの結果の幾何平均である．各折れ線の“RMT-Nw”は，1サイクルあたりN命令（$N \times 2$オペランド）のリネーミングができることを表す．同図より，RMTの処理幅が2命令以上ないと10 %以上性能が低下してしまうことがわかる．また，RMTの処理幅が2命令の場合，一致経路長は6命令のときが最も性能が良い．このため，以降ではRMTの処理幅を2命令，一致経路長を6命令として評価を行う．

5.4 IPC

図 12に，各モデルのBaseに対する相対IPCを示す．RTCPropはBaseに対し，平均で0.3%性能を低

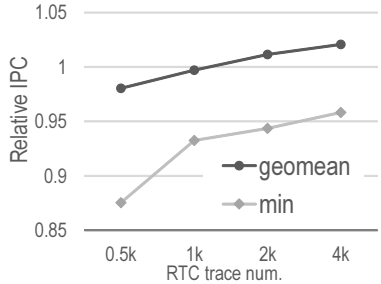

図 10 容量を変化させた場合の相対 IPC

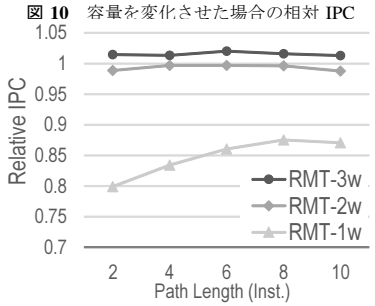

図 11 一致経路長と RMT の幅を変化させた場合の相対 IPC

☆ これらはタグ部分までを含んだ場合の容量である．

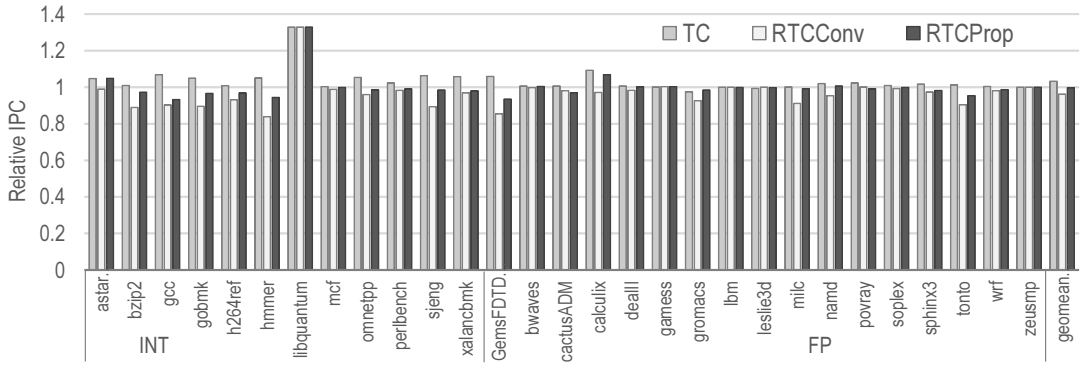

図 12 Base に対する相対 IPC

下させている．4.3.1 節で述べたように提案手法では容量効率を改善しているため，3.1 節で述べた問題による性能低下は，TC 方式と同様のフェッチ・バンド幅拡大による性能向上によって相殺できる程度となっている．これに対し，TC と比較した場合はフェッチ・バンド幅拡大の効果は同様であるため，平均で 3.4%性能を低下させている．

RTCConv は，Base とくらべて平均で 3.8%性能を低下させている．また，RTCConv では大きく性能を下げているベンチマークがあり，特に hmmer や GemsFDTD，sjeng ではそれぞれ 16%，15%，11% 性能を低下させている．これに対し，RTCProp の性能低下は最悪の場合でも GemsFDTD の 7.6%であり，RTCConv よりも大きく性能を改善している．

5.5 消費エネルギー

図 13 に，各モデルの Base に対するオーバーヘッドを含んだ RMT の相対消費エネルギーを示す．同図の結果は，全ベンチマークの結果を平均したものである．同図内の overhead は，フェッチ部分で生じる消費エネルギーのオーバーヘッドを表す．TC，RTCConv，RTCProp では，TC や RTC，および命令キャッシュにアクセスが行われる．これらによる消費エネルギーは Base の命令キャッシュ・アクセスによる消費エネルギーよりも大きいため，それにより増加した消費エネルギーを同図では overhead として表している．

RTCProp の消費エネルギーは，Base，TC，RTCConv と比較してそれぞれ 53%，56%，64%削減されている（オーバーヘッド分を含む）．これは RMT の消費エネルギーが大きく削減されているためであり，オーバーヘッドを含まない RTCProp の RMT による消費エネルギーは，Base の RMT の 18%にまで削減されている．RTCProp と同様に RTCConv では RMT の消費エネルギーは大きく削減されているものの，オーバーヘッドが大きく，これにより逆に消費エネルギーを全体で 32%増加させている．これは 3.2 節で述べた，間接分岐ターゲット・アドレスをタグに格納していることによるハードウェア増加の影響によるものである．RTCProp ではタグ部分に間接分岐ターゲット・アドレスを含まないため，RTC の消費エネルギーは RTCConv と比べて大幅に小さく，オーバーヘッドも全体として消費エネルギーを削減できる程度まで縮小されている．

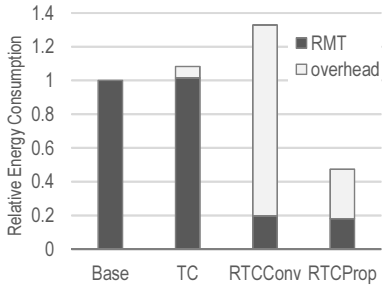

図 13 相対消費エネルギー

6. 関 連 研 究

RMT の持つ問題の解決を目的として，RMT に小容量の **Register Map Cache**（**RMC**）を追加する手法が提案されている[11),12)]．RMC は，**Main Register Map Table**（**MRMT**）の一部を保持するキャッシュである．アクセスの大部分を RMC に集中させることにより，MRMT のポート数を削減させる．

Liu らは，RMT のアクセス・レイテンシを短縮するために RMC を用いる手法を提案している[11)]．この手法では，通常のキャッシュと同様に RMC のヒット時にはレイテンシを短縮する．RMC のミス時には，フロントエンドをストールさせ，その間に MRMT に対してアクセスする．Liu らの手法では，RMC のペナルティによって性能が大きく下がってしまう場合があった．

これに対し，三輪らは**ミスを仮定したパイプライン**[13)] を用いることにより，RMC による性能の低下を避けながら RMT の面積を削減する手法を提案している[12)]．三輪らの手法では，性能をほとんど落とすことなく，大幅に RMT の回路面積を縮小することができる．しかし，三輪らの手法では，MRMT のポート数は削減されるものの，RMC のポート数が削減されな

い．このため，リネーム幅を増加させた場合には，この RMC の回路面積と消費エネルギーが問題となる．

これらの RMC を用いる手法は，提案手法の RMT にも適用可能であり，それによって消費エネルギーをさらに下げられることが期待できる．

7. おわりに

Out-of-order スーパスカラ・プロセッサのリネーム・ロジックは非常に大きな電力を消費する．これを解決するために我々はこれまでに RTC を提案してきたが，キャッシュの容量効率低下やタグ部分のハードウェア量増加により，性能や消費エネルギーに問題があった．これに対し，本論文では RTC に格納するオペランドのリネーミング結果を，そのプロデューサまでの距離が近いものに制限する手法を提案した．5 節の評価では，従来の命令キャッシュを持つプロセッサと比較して，性能低下を 0.3%にとどめながら，リネーミングに関わる消費エネルギーを 53%削減できることを確かめた．今後は McPAT[14] などのプロセッサ全体の消費エネルギーを評価できるシミュレータを使用し，プロセッサ全体に対する消費エネルギー削減の効果を評価する予定である．

謝　辞

本研究の一部は，日本学術振興会 科学研究費補助金 若手研究 (A)（課題番号 24680005）による補助のもとで行われた．

参考文献

1) 一林宏憲, 塩谷亮太, 入江英嗣, 五島正裕, 坂井修一: 逆 Dualflow アーキテクチャ, 情報処理学会論文誌コンピューティングシステム ACS, Vol. 1, No. 2, pp. 22–33 (2008).
2) Rotenberg, E., Bennett, S. and Smith, J. E.: Trace cache: a low latency approach to high bandwidth instruction fetching, *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 24–35 (1996).
3) Rixner, S., Dally, W., Khailany, B., Mattson, P., Kapasi, U. and Owens, J.: Register organization for media processing, *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 375–386 (2000).
4) Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28–41 (1996).
5) Preston, R., Badeau, R., Bailey, D., Bell, S., Biro, L., Bowhill, W., Dever, D., Felix, S., Gammack, R., Germini, V., Gowan, M., Gronowski, P., Jackson, D., Mehta, S., Morton, S., Pickholtz, J., Reilly, M. and Smith, M.: Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading, *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, Vol. 1, pp. 334–472 (2002).
6) Safi, E., Moshovos, A. and Veneris, A.: On the latency and energy of checkpointed superscalar register alias tables, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 18, No. 3, pp. 365–377 (2010).
7) Sinharoy, B., Kalla, R., Starke, W. J., Le, H. Q., Cargnoni, R., Van Norstrand, J. A., Ronchetti, B. J., Stuecheli, J., Leenstra, J., Guthrie, G. L., Nguyen, D. Q., Blaner, B., Marino, C. F., Retter, E. and Williams, P.: IBM POWER7 Multicore Server Processor, *IBM J. Res. Dev.*, Vol. 55, No. 3, pp. 191–219 (2011).
8) The Standard Performance Evaluation Corporation: *SPEC CPU2006 suite* `http://www.spec.org/cpu2006/`.
9) 塩谷亮太, 五島正裕, 坂井修一: プロセッサ・シミュレータ「鬼斬弐」の設計と実装, 先進的計算基盤システムシンポジウム SACSIS, pp. 120–121 (2009).
10) Muralimanohar, N., Balasubramonian, R. and Jouppi, N.: CACTI 6.0: A tool to model large caches, Technical report, HP Laboratories (2009).
11) Liu, T. and Lu, S.-L.: Performance improvement with circuit-level speculation, *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 348–355 (2000).
12) 三輪忍, 張鵬, 横山弘基, 堀部悠平, 中條拓伯: キャッシュを用いたレジスタ・マップ表の回路面積削減, 情報処理学会論文誌コンピューティングシステム ACS, Vol. 3, No. 3, pp. 44–55 (2010).
13) Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register Cache System not for Latency Reduction Purpose, *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 301–312 (2010).
14) Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M. and Jouppi, N. P.: McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures, *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 469–480 (2009).