Tag Comparison Omitting for Low-Power Instruction Caches

井上, 弘士
九州大学大学院システム情報科学研究科情報工学専攻

村上, 和彰
九州大学大学院システム情報科学研究院

出版情報：情報処理学会研究報告. 計算機アーキテクチャ研究会報告. 2000 (110), pp.25-30, 2000-11-29. Information Processing Society of Japan
バージョン：
権利関係：

実行履歴に基づいた低電力命令キャッシュ向けタグ比較回数削減手法

井上 弘士 †　　村上 和彰 ‡

† 九州大学 大学院システム情報科学研究科 情報工学専攻

‡ 九州大学 大学院システム情報科学研究院

〒 816-8580 福岡県春日市春日公園 6-1

E-mail: *ppram@c.csce.kyushu-u.ac.jp*

あらまし：現在多くのプロセッサ・チップには，当然のようにキャッシュ・メモリが搭載されている．また，更なるヒット率の向上を目的として，キャッシュ・サイズは年々増加傾向にある．そのため，キャッシュ・アクセスにおける消費エネルギーが増大し，ひいては，チップの全消費エネルギーに大きな影響を与えるようになってきた．特に，命令キャッシュへのアクセスは毎クロック・サイクル発生するため，その低消費エネルギー化が極めて重要となる．そこで本稿では，ダイレクト・マップ命令キャッシュの低消費エネルギー化を目的として，ヒストリ・ベース・タグ比較方式を提案する．プログラムの実行履歴に基づき，必要に応じてタグ比較を行うことで，キャッシュ・アクセス当りの消費エネルギーを削減できる．複数ベンチマークを用いた実験の結果，従来型タグ比較方式と比較して，全ての浮動小数点プログラムで約 90%以上，2 つの整数プログラムで約 80%以上のタグ比較を削減できた．

キーワード　低消費電力，キャッシュ，タグ比較，実行履歴，動的最適化

Tag Comparison Omitting for Low-Power Instruction Caches

Koji Inoue †　　Kazuaki Murakami ‡

† Department of Computer Science and Communication Engineering, Kyushu University

‡ Department of Informatics, Kushu University

E-mail: *ppram@c.csce.kyushu-u.ac.jp*

Abstract : Almost all microprocessor chips employ on-chip caches, because confining memory accesses in on-chip reduces the latency for memory accesses and the frequency of driving high capacitance I/O pads. The trend increases the cache capacity to confine more memory accesses in on-chip for achieving high performance and low power consumption. However, this trend also increases the energy consumed for cache accesses. In this paper, we propose a novel architecture for low-power direct-mapped instruction caches, called *history-based tag-comparison cache*. The cache attemps to reduce the count of tag comparison required for program execution, thereby saving the energy. Program-execution history recorded in a BTB (Branch Target Buffer) is used for the tag-comparison omittion. In our evaluation, it is observed that more than 90 % and 80 % of tag-comparison count are reduced for floating-point programs and some integer programs, respectively.

key words　low power, cache, tag comparison, execution history, run-time optimization

1 はじめに

急速な携帯電子機器市場の拡大を背景に,「性能とコストの時代」から,「性能とコストと消費エネルギー (消費電力) の時代」へと突入した．また，超高性能化を第一目的としたスーパコンピュータの分野においても，低消費電力化が重要視されるようになってきた．発熱の問題により動作速度の向上が抑制されるためである．一方，携帯電子機器といえども，実行対象となるアプリケーションはより高機能化している．そのため，低消費エネルギー化 (または，低消費電力化) と高性能化といった相反する要求を同時に達成する必要がある．

現在，多くのプロセッサ・チップには，当然のようにオンチップ・キャッシュ(以下，キャッシュ) が搭載されている．オフチップ・メモリアクセス回数を削減することで，1) 平均メモリアクセス時間の短縮による高性能化，2) 外部 I/O ピン駆動頻度の低減による低消費エネルギー化，を達成できるためである．しかしながら，更なるヒット率の向上を目的として，キャッシュ・サイズは年々増加傾向にある．その結果，キャッシュ・アクセス当りの消費エネルギーが増大し，チップの全消費エネルギーに大きな影響を与えるようになってきた [3]．特に，命令キャッシュへのアクセスは毎クロック・サイクル発生するため，その低消費エネルギー化が極めて重要となる．

そこで本稿では，ダイレクト・マップ命令キャッシュの低消費エネルギー化を目的として，ヒストリ・ベース・タグ比較方式を提案する．また，ベンチマーク・プログラムを用いた定量的評価を行い，その有効性を明らかにする．通常，全てのキャッシュ・アクセスにおいて，参照データがキャッシュ中に存在するか否かを調べるためにタグ比較を行う必要がある．これに対し，ヒストリ・ベース・タグ比較方式では，プログラムの実行履歴に基づき，必要に応じてタグ比較を行う．そして，プログラム実行時の総タグ比較回数を削減することで，命令キャッシュの低消費エネルギー化を実現する．

以下，第 2 章では，キャッシュ・アクセスに関する消費エネルギー・モデルを定義する．また，タグ比較における低消費エネルギー化の重要性を示す．次に第 3 章では，比較対象として，過去に提案されたインターライン・タグ比較方式を説明する．そして，第 4 章では我々が提案するヒストリ・ベース・タグ比較方式の詳細を示し，第 5 章でベンチマーク・プログラムを用いた定量的評価を行う．最後に，第 6 章で簡単にまとめる．

2 キャッシュの消費エネルギー

2.1 消費エネルギー・モデル

キャッシュ・アクセスにおいて消費されるエネルギー $Ecache$ は，主に，1) アドレス・デコードに要するエネルギー E_{decode}，2)SRAM セルへのアクセスに要するエネルギー E_{sram}，3) 外部入出力ピン駆動に要するエネルギー E_{io}，の総和で近似できる [6].

$$E_{cache} \quad = \quad E_{decode} + E_{sram} + E_{io} \qquad (1)$$

ここで，E_{cache} を構成する各要素について考察する．E_{decode} は，E_{sram} や E_{io} に比べ，E_{cache} に与える影響が極めて小さいと報告されている [1]．一方，外部入出力ピンの駆動はライン・リプレイス時に発生するため，E_{io} はキャッシュ・ヒット率に依存する．一般的に，命令参照は非常に高い局所性を有しており，多くのプログラムで高いヒット率を達成できる．また，今後の SOC(System-On-a-Chip) においてキャッシュと主記憶が 1 チップ化された場合，外部入出力ピンの駆動を伴うことなくライン・リプレイスを行える．そこで本稿では，E_{io} が E_{cache} に与える影響も小さいと仮定し，キャッシュ・アクセスにおける消費エネルギーを以下のように E_{sram} で近似する．

$$E_{cache} \quad = \quad E_{sram} = E_{sram\_tag} + E_{sram\_data} \qquad (2)$$

ここで，E_{sram_tag} ならびに E_{sram_data} は，それぞれ，キャッシュ・アクセスにおけるタグメモリ・アクセスならびにデータメモリ・アクセスで消費されるエネルギーである．なお，データメモリとは，キャッシュ・ラインを格納するための SRAM アレイである．

2.2 タグ比較における消費エネルギー

通常，1 つのキャッシュ・ラインには複数のワード・データが格納されており，これらが同時にデータメモリから読出される．そのため，データメモリへのアクセスでは，キャッシュ・ライン・サイズに相当する数のビット線が充放電の対象となる．これに対し，タグメモリへのアクセスでは，命令アドレスのタグフィールドに相当する数のビット線が充放電の対象となる．例えば，32 ビット語長 CPU において，32 バイトのラインサイズを有する 32K バイト・ダイレクトマップ命令キャッシュの場合，ラインサイズが 256 ビット (32×8) であるのに対し，タグサイズは 17 ビット (32 ビット語長 - 10 ビットのインデックス - 5 ビットのオフセット) となる．したがって，第 2.1 節で示した式 (2) は，データメモリ・アクセスにおける消費エネルギー (E_{sram_data}) に大きく依存する．

データメモリ・アクセスにおける消費エネルギーは，キャッシュ・サブバンキングを行うことで大幅に削減できる [3]．サブバンキング技術を活用した 32K バイト・ダイレクトマップ命令キャッシュのアクセス消費エネルギーを図 1 に示す[1]．横軸はサブバンク数，縦軸は命令キャッシュ･アクセス当りの平均消費エネルギーを示す．なお，全ての結果は，サブバンク数が 1 である従来型キャッシュの消費エネルギーに正規化している．E_{bit_tag} および E_{bit_data} は，それぞれ，タグメモリおよびデータメモリにおけるビット線の充放電で消費されるエネルギーである．

32 ビット語長の場合，サブバンク構成を採らない従来型キャッシュでは，タグメモリ・アクセスに要するエネルギーは全体の約 6%程度である．しかしながら，サブバンク数を増やすにつれ，キャッシュ・アクセス当りの消費エネルギーが減少すると共に，タグメモリ・アクセスに要する消費エネルギーの割合が大きくなる．サブバンク数が 8 の場合，全体の約 33%のエネルギーをタグメモリ・アクセスによって消費する．また，64 ビット語長の場合には，40%以上のエネルギーを消費することになる．したがって，より低消費電力なキャッシュを実現するためには，データメモリ・アクセスだけでなく，タグメモリ・アクセスにおける消費電力の削減も重要となる．

3 従来型条件付きタグ比較 (インターライン・タグ比較方式)

本節では，従来型条件付きタグ比較方式として，文献 [5] で提案されたインターライン・タグ比較方式を説明する．命令 i と j が連続して実行される時，これら命令間の実行フロー関係は以下のように分類できる．

[1] 図 1 における消費エネルギーは，文献 [3] の計算式を用いて求めた．また，計算式中にて使用する負荷容量等の各種パラメータに関しては，文献 [4] の値を参照した．

図 1: サブバンキングにより低消費エネルギー化

- イントラライン・シーケンシャル・フロー：命令 i と j のアドレスは**連続**しており，かつ，これらは**同一キャッシュ・ライン**内に存在する．
- イントラライン・ノンシーケンシャル・フロー：命令 i と j のアドレスは**非連続**であり，かつ，これらは**同一キャッシュ・ライン**内に存在する．この場合，命令 i は分岐命令である．
- インターライン・シーケンシャル・フロー：命令 i と j のアドレスは**連続**しており，かつ，これらは**異なるキャッシュ・ライン**内に存在する．この場合，キャッシュ・ライン境界は命令 i と j の間に存在する．
- インターライン・ノンシーケンシャル・フロー：命令 i と j のアドレスは**非連続**であり，かつ，これらは**異なるキャッシュ・ライン**内に存在する．この場合，命令 i は分岐命令である．

通常，あるメモリ参照においてキャッシュ・ミスが発生した時，参照データと同一キャッシュ・ライン内に存在する全てのデータがリフィルされる．よって，連続する 2 つの命令が，イントラライン・シーケンシャル・フローまたはイントラライン・ノンシーケンシャル・フローの関係にある場合，命令 j におけるキャッシュ・アクセスは必ずヒットする．そこで，プログラムカウンタの値を監視してインターライン・フローを検出し，その場合にのみタグ比較を行うことで総タグ比較回数を削減できる．

4 ヒストリ・ベース・タグ比較方式

4.1 コンセプト

まず，用語の定義を行う．本稿では，連続実行される命令列を**動的基本ブロック**と呼ぶ．動的基本ブロックは，1 個またはそれ以上の基本ブロック (ここでは，動的基本ブロックと区別するために**静的基本ブロック**と呼ぶ) で構成される．ある 1 つの動的基本ブロックが複数の静的基本ブロックで構成される場合，当該動的基本ブロック内には不成立となった複数個の条件分岐命令が存在する．したがって，動的基本ブロックの大きさは，これら条件付き分岐命令に依存しており，プログラム実行時に変化する．

一般に，命令参照は非常に高い局所性を有しており，多くのプログラムで高ヒット率を達成できる．しかしながら，従来の命令キャッシュでは，ほとんどのアクセスがヒットするにも関わらず，毎回タグ比較を行わなければならない．例えば，命令キャッシュのヒット率が 98%の場合，2%のキャッシュ・ミスを検出するために，98%のタグ比較も行う必要がある．もし，タグ比較を行うことなしにキャッシュ・ヒットもしくはミスを判定できれば，タグ比較処理を省略できる．

ここで，プログラムの特性ならびにキャッシュ・メモリの動作に起因する以下の事実に着目する．

- 多くのプログラムはループ構造に基づく．したがって，ある命令列が繰り返し参照 (実行) される確率が高い．
- キャッシュの内容が変更されるのは，キャッシュ・ミスが発生し，新しいデータがリフィルされる時 (または，あるデータが追い出される時) である．つまり，高ヒット率を達成できる命令キャッシュにおいて，その内容は極めて稀にしか更新されない．

このような事実を利用して，参照すべき命令がキャッシュ中に滞在しているか否かをキャッシュ・アクセス開始前に (タグ比較を行うことなしに) 判定できる．例えば，あるループ内に存在する動的基本ブロックの実行について考える．最初にこの動的基本ブロックを実行する時，キャッシュ・ミスが発生する可能性があるため，必ずタグ比較処理を行う必要がある．しかしながら，次のイタレーションにおいて，前回の実行から現在に至るまで一度もキャッシュ・ミスが発生していなければ (つまり，キャッシュの内容が更新されていなければ)，この動的基本ブロックはキャッシュ内に滞在していることが保証される．よって，当該動的基本ブロックの 2 回目の実行に関しては，命令キャッシュ・アクセスにおけるタグ比較処理を省略可能となる．ヒストリ・ベース・タグ比較方式では，ある動的基本ブロックの実行において，以下の条件を満足する時にタグ比較処理を省略し，低消費エネルギー化を実現する．

- 現在実行対象である動的基本ブロックは過去に実行された実績があり，かつ，
- 前回の実行から現在に至るまで一度もキャッシュ・ミスが発生していない．

実際には，プログラムの実行履歴として「実行足跡 (execution footprints)」を記録する．ある動的基本ブロックを実行した時，対応する実行足跡を残す．全ての実行足跡は，キャッシュ・ミスの発生と同時に消去される．その後，当該動的基本ブロックが再度実行される際，今だ実行足跡が残っていれば，全ての命令列はキャッシュ内に存在すると判断し，タグ比較処理を省略する．第 3 で示したインターライン・タグ比較方式との違いは，イントラライン・フローだけでなく，インターライン・フローの場合でもタグ比較を省略可能な点である．

4.2 内部構成

第 4.1 節で述べたように，ヒストリ・ベース・タグ比較方式では，動的基本ブロックの実行足跡を記録する必要がある．動的基本ブロックが複数の条件分岐命令を含む場合，その大きさはこれら条件分岐命令の実行結果に依存する．したがって，動的基本ブロックの実行足跡は，その有効範囲 (つまり，当該動的基本ブロックの先頭アドレスならびに最終アドレス) に関する情報を保持していなければならない．そこで，実行足跡の記録には，分岐

図 2: BTB の構成

予測機構における BTB(Branch Target Buffer) を利用する．近年の多くの商用マイクロプロセッサは，分岐ペナルティの削減を目的として，分岐予測機構を搭載している．主に，分岐予測機構は，分岐条件を予測するための BPT(Branch Prediction Table) と，分岐先アドレスを予測するための BTB で構成する．BTB の各エントリには，分岐アドレス・フィールドと分岐先アドレス・フィールドがある．PC(Program Counter) の値と分岐アドレス・フィールドの値が一致し，かつ，BPT による分岐予測結果が成立 (taken) であれば，分岐先アドレス・フィールドの値が PC に設定される．また，BTB に未登録である分岐命令の実行結果が成立 (taken) であった場合には，当該分岐命令のアドレスならびに分岐先アドレスが新しいエントリとして BTB に登録される．

ヒストリ・ベース・タグ比較方式における BTB の構成を図 2 に示す．BTB エントリにおいて，その分岐先アドレス・フィールドは，ある動的基本ブロックの先頭アドレスを示す．一方，分岐アドレス・フィールドは，ある動的基本ブロックの先頭アドレスとなる．ヒストリ・ベース・タグ比較方式では，従来型 BTB の構成に以下のハードウェア機構を設ける．

- *RCT (Residing in the Cache on Taken)* フラグ：各エントリに実装された 1 ビットのフラグであり，同一エントリの分岐先アドレスを先頭とする動的基本ブロックの実行足跡を示す．同一エントリ内の分岐アドレスと PC が一致し，かつ，当該分岐命令の分岐予測が成立であった場合，このフラグは 1 にセットされる (つまり，実行足跡を残す)．一方，キャッシュ・ミスが発生した時には 0 にリセットされる (つまり，実行足跡を消去する)．したがって，1 にセットされた RCT フラグは，分岐先アドレスを先頭とする動的基本ブロックに関して，過去に実行された実績が有り，かつ，現在もキャッシュ中に存在することを示す．

- *RCN (Residing in the Cache on Not-taken)* フラグ：各エントリに実装された 1 ビットのフラグであり，同一エントリにおける分岐アドレスの次アドレスを先頭する動的基本ブロック (実際には，実行中の動的基本ブロックにおける残りの命令列) の実行足跡を示す．同一エントリ内の分岐アドレスと PC が一致し，かつ，当該分岐命令の分岐予測が不成立であった場合，このフラグは 1 にセットされる．一方，キャッシュ・ミスが発生した時には 0 にリセットされる．したがって，1 にセットされた RCN フラグは，当該分岐命令以降の連続した命令列に関して，過去に実行された実績が有り，かつ，現在もキャッシュ中に存在することを示す．

図 3: BTB の基本動作

また，キャッシュ・アクセスにおけるタグ比較処理を省略するため，以下のフラグを設ける．

- *TCO (Tag Comparison Omitting)* フラグ：タグ比較を省略可能か否かを示す 1 ビットのフラグである．このフラグが 1 の時，命令キャッシュ・アクセスにおけるタグ比較処理は省略される．

4.3 動作

4.3.1 基本動作

実行足跡 (RCT および RCN フラグ) は，プログラム実行フローに基づいてセットされ，キャッシュ・ミスが発生した際にリセットされる．これに加え，BTB エントリのリプレイスが発生した場合にも実行足跡をリセットする．これは，BTB エントリが追い出された場合，ある動的基本ブロックの最終アドレス情報が失われるためである．BTB アクセス時の動作フローを図 3(A) に示す．以下，各状態における動作を説明する．

1. キャッシュ・ミスが検出された場合，全ての足跡を消去する．つまり，全ての RCT および RCN フラグを 0 にリセットする．なお，TCO フラグが 1 の場合，キャッシュ・アクセスは必ずヒットするため，本状態は無視され，状態 2 へと遷移する．

2. 現在の PC に対応する BTB エントリが存在しない場合，後続命令列が実行される．したがって，本動作フローは終了し，次の命令フェッチにそなえて初期状態 (図中の Start) に遷移する．

3. RAS(Return Address Stack) を利用することで，より正確な分岐予測を行える [2]．しかしながら，本稿では RAS に対する実行履歴の記録は考慮していない．そのため，RAS から分岐先アドレスを得た場合には，無条件で TCO フラグを 0 にリセットする (タグ比較処理の開始する)．

4. 実行足跡に従って TCO フラグを設定する．もし，分岐予測結果が成立であれば，現在の PC に対応する BTB エントリ内の RCT フラグを TCO フラグにコピーする．また，実行足跡として当該 RCT フラグを 1 にセットする．一方，分岐予測結果が不成

立の場合には，対応する RCN フラグを TCO フラグにコピーし，実行足跡として当該 RCN フラグを 1 にセットする．

分岐予測が外れた場合，プロセッサは正しい分岐条件に基づき PC の値を修正しなければならない．この時，該当する BTB エントリが更新される可能性がある．誤った分岐予測によって生じる PC 修正時の動作を図 3(B) に示す．

5. BTB の更新は分岐成立時にのみ発生し，1) 全く新しいエントリを追加する場合と，2) 現在登録済みのエントリにおいて分岐先アドレスのみを更新する場合がある．前者において，もし新規に追加されるエントリが他エントリを追い出す場合，全ての実行足跡を消去する．一方，後者の場合，分岐先アドレス・フィールドの値が変更されるだけである．つまり，ある動的基本ブロックの最終アドレスを示す分岐アドレスは保持される．そのため，実行足跡の消去を行う必要は無い．その後，両者において，TCO フラグが 0 にリセットされ (つまり，タグ比較処理の開始)，実行足跡として RCT フラグを 1 にセットする．

6. 正しい分岐条件に従って，TCO フラグの値を RCT もしくは RCN フラグにコピー・バックする．また，再度 TCO フラグの設定を行い，正しい実行足跡を記録する．

4.3.2 動作例

あるプログラム実行における BTB の動作例を図 4 に示す．図中 (A) は，7 回の繰返しを有するループ文実行における制御フローである．実線矢印は各イタレーションでの動的基本ブロック (先頭アドレスは A) を，また，破線矢印は分岐先を表す．この動的基本ブロックのサイズは，アドレス B，C，ならびに，D で示される条件分岐命令の実行結果に依存する．一方，図中 (B) は，プログラム実行における BTB の内部状態 (RCT フラグと RCN フラグ)，ならびに，TCO フラグの値を示している．数字と英大文字の組は時刻を表す．例えば，$1-C$ とは，イタレーション 1 においてアドレス C の条件分岐命令が実行された時刻である．以下，プログラム実行に伴う BTB 内部状態および TCO フラグの変化を説明する．なお，本例では，分岐予測結果は全て正しいと仮定する．

1-C ： 条件分岐-C(アドレス C) が BTB に登録される．そのため，TCO フラグは 0 にリセットされ，タグ比較処理を開始する．また，分岐条件結果に基づき，実行足跡として RCT フラグが 1 にセットされる．

2-C ：分岐条件結果は成立である．したがって，時刻 1-C で残した RCT フラグが TCO フラグに設定され，タグ比較処理を中断する．

4-C ： 条件分岐-C の条件が不成立となり，動的基本ブロックのサイズは，前イタレーション実行時と比較して大きくなる．この場合，RCN フラグが TCO フラグに設定され，タグ比較処理を再開する．また，RCN フラグが実行足跡として 1 にセットされる．

図 4: 動作例

4-D ：条件分岐-D(アドレス D) が初めて実行され，BTB に登録される．TCO フラグは 0 のままであり，タグ比較処理は継続される．また，対応する RCT フラグが実行足跡として 1 にセットされる．

5-C ：条件分岐-C は BTB に登録済みのため，この時刻において BTB アクセスが発生する．ここでは，分岐条件は不成立である．よって，時刻 $4-C$ で残した RCN フラグが TCO に設定され，タグ比較処理を中断する．

7-B ：条件分岐-B(アドレス B) の条件が成立となり BTB に登録される．この時，前イタレーション実行時と比較して，動的基本ブロックのサイズは小さくなる．TCO フラグが 0 にリセットされ，タグ比較処理を再開する．また，実行足跡として，対応する RCT フラグが 1 にセットされる．

5 評価

5.1 実験環境

イベント・ドリブン・シミュレーションにより，プログラム実行時に発生する総タグ比較回数を測定した．ベンチマーク・プログラムとしては SPEC95 ベンチマーク・サイトから，8 個の整数プログラム (*099.go*, *124.m88ksim*, *129.compress*, *126.gcc*, *130.li*, *132.ijpeg*, *134.perl*, *147.vortex*)，および，5 個の浮動小数点プログラム (*102.swim*, *107.mgrid*, *110.applu*, *125.turb3d*, *141.apsi*) を利用した．なお，整数プログラムに関しては train 入力データを，浮動小数点プログラムに関しては test 入力データを用いた．実際には，SimpleScalar シミュレーション・ツール・セット (リリース 2.0)[7] における BTB シミュレータを拡張し，以下の比較対象モデルに関して総タグ比較回数を求めた．

- **C-TC** (Conventional Tag-Comparison)： 従来型キャッシュの構成であり，命令フェッチ毎にタグ比較を行う．
- **IL-TC** (InterLine Tag-Comparison)： 第 3 章で説明したインターライン・タグ比較方式であり，インターライン・フローが発生した時のみタグ比較を行う．
- **H-TC** (History-based Tag-Comparison)： 第 4 章で述べたヒストリ・ベース・タグ比較方式であり，TCO フラグの値に従ってタグ比較を行う．

表 1: Normalized Tag-Comparison Counts

Benchmark	C-TC	IL-TC	H-TC	H-TC ideal	HIL TC
099.go	1.000	0.3203	0.7604	0.4027	0.2378
124.m88ksim	1.000	0.3302	0.4217	0.1856	0.1361
129.compress	1.000	0.3528	0.1751	0.1718	0.0706
126.gcc	1.000	0.3343	0.6810	0.2812	0.2278
130.li	1.000	0.3515	0.4500	0.1811	0.1684
132.ijpeg	1.000	0.2992	0.1062	0.0560	0.0311
134.perl	1.000	0.3436	0.6643	0.1361	0.2249
147.vortex	1.000	0.3213	0.8838	0.2141	0.2837
102.swim	1.000	0.2957	0.0623	0.0622	0.0278
107.mgrid	1.000	0.2600	0.0008	0.0002	0.0002
110.applu	1.000	0.2657	0.0252	0.0248	0.0070
125.turb3d	1.000	0.2813	0.0849	0.0727	0.0266
141.apsi	1.000	0.2801	0.1050	0.0476	0.0307

- **H-TC ideal** : ハードウェア制約を考慮しない理想的なヒストリ・ベース・タグ比較方式．H-TC との違いは，パーフェクト命令キャッシュ(キャッシュ・ミスは発生しない)，ならびに，フルアソシアティブ BTB(BTB における競合ミスが発生しない) を有する点である．
- **HIL-TC** (History-based InterLine Tag-Comparison) : インターライン・タグ比較方式 (IL-TC) とヒストリ・ベース・タグ比較方式 (H-TC) の組合わせ．TCO フラグが 0 であり，かつ，インターライン・フローの場合にのみタグ比較を行う．

なお，H-TC ideal のキャッシュ・サイズならびに BTB 連想度を除き，全ての評価モデルにおいて，命令キャッシュ・サイズは 32K バイト，ラインサイズは 32 バイト，分岐条件予測機構は 2 ビット・カウンタ，BPT エントリ数は 2048，BTB エントリ数は 2048(セット数 512 の 4 ウェイ)，RAS サイズは 8 と仮定する．

5.2 実験結果

シミュレーション結果を表 1 に示す．各プログラムにおいて，全ての結果は従来方式 (C-TC) の結果に正規化している．まず，インターライン・タグ比較方式 (IL-TC) とヒストリ・ベース・タグ比較方式 (H-TC) を比較する．一般に，プログラムの大部分はインクリメンタル実行である．そのため，インターライン・タグ比較方式は，全てのプログラムにおいてタグ比較回数を 65〜70%削減している．これに対し，ヒストリ・ベース・タグ比較方式は，プログラムが有する繰返し実行性 (ループ構造) を活用するため，その度合いによって効果は様々である．例えば，全ての浮動小数点プログラムに関しては 90〜99%，*129.compress* や *132.ijpeg* の整数プログラムにおいては 80〜90% のタグ比較回数削減を達成した．これらのプログラムは，比較的固定のループ構造を有しているためと考える．これに対し，*147.vortex* や *099.go* といった整数プログラムでは，約 10〜25%程度のタグ比較回数削減率であった．

次に，ハードウェア制約を無視したヒストリ・ベース・タグ比較方式の理想モデル (H-TCideal) と，現実的モデル (H-TC) を比較する．前述したように，現実的モデルでは幾つかの整数プログラムに対して，大きな削減効果を得ることはできなかった．しかしながら，理想モデルではこれら整数プログラムに対しても大きなタグ比較回数削減率を達成している．例えば，*147.vortex* のタグ比較削減率は，現実モデルが約 12%であるのに対し，理想モデルでは約 80% である．この結果より，ヒストリ・ベース・タグ比較方式の効果は，命令キャッシュおよび BTB のハードウェア制約による影響が大きいと考える．

最後に，従来方式 (C-TC) と組合わせ方式 (HIL-TC) を比較する．全ての浮動小数点プログラムにおいて，組合わせ方式では約 97%以上のタグ比較回数削減を達成した．また，整数プログラムに関しても，タグ比較回数削減率は最高で約 97%(*132.ijpeg*)，最低で約 72%(*147.vortex*) であった．このように，インターライン・タグ比較方式とヒストリ・ベース・タグ比較方式の組み合わせにより，極めて大きなタグ比較回数削減効果を得ることができた．

6 おわりに

本稿では，ダイレクト・マップ命令キャッシュ用低消費電力化手法として，ヒストリ・ベース・タグ比較方式を提案した．本方式では，実行履歴に基づき，アクセス対象命令がキャッシュ内に存在するか否かを予め判定する．これにより，タグ比較の実行を省略し，タグメモリ・アクセスならびにタグ比較における消費エネルギーを削減できる．

複数ベンチマークを用いて実験を行った結果，従来タグ比較方式と比較して，全ての浮動小数点プログラムで約 90〜99%，2 つの整数プログラムで約 80%以上のタグ比較回数削減を達成した．また，過去に提案されたインターライン・タグ比較方式との組合わせにより，さらなるタグ比較回数の削減が可能であることを示した．今後，パイプライン構造を考慮したタイミング・シミュレーションに基づき，より詳細な評価を行う予定である．

謝辞

日頃から御討論頂く，九州大学 大学院システム情報科学研究院 安浦寛人教授，岩井原瑞穂 助教授，PPRAM グループ関係者各位，ならびに，研究室の諸氏に感謝します．なお，本研究は一部，文部省科学研究費補助金基盤研究 (A)(2) 展開研究「システム LSI 向きカスタム化可能 IP コアのアーキテクチャおよび設計支援技術の開発」(課題番号：12358002)，展開研究「メモリ/ロジック混載技術に基づく大規模集積回路システム・アーキテクチャの研究開発」 (課題番号：09358005)，ならびに，一般研究「スケーラブル・システム LSI アーキテクチャの設計手法に関する研究」(課題番号：11308011) による．

参考文献

[1] R. I. Bahar, G. Albera, and S. Manne, “Power and Performance Tradeoffs using Various Caching Strategies,” *Proc. of the 1998 International Symposium on Low Power Electronics and Design*, pp.64–69, Aug. 1998.

[2] D. R. Kaeli, and P. G. Emma, “Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns,” *Proc. of the 18th Annual International Symposium on Computer Architecture*, pp.34–42, May 1991.

[3] M. B. Kamble, and K. Ghose,“Analytical Energy Dissipation Models For Low Power Caches,” *Proc. of the 1997 International Symposium on Low Power Electronics and Design*, pp.143–148, Aug. 1997.

[4] M. B. Kamble, and K. Ghose,“Energy-Efficiency of VLSI Caches: A Comparative Study,” *Proc. of 10th International Conference on VLSI Design*, pp.261–267, Jan. 1997.

[5] R. Panwar, and D. Rennels, “Reducing The Frequency of Tag Compares for Low Power I-cache Design,” *Proc. of the 1995 International Symposium on Low Power Electronics and Design*, pp.57–62, Apr. 1995.

[6] C. L. Su, and A. M. Despain, “Cache Design Trade-offs for Power and Performance Optimization:A Case Study,” *Proc. of the 1995 International Symposium on Low Power Design*, pp.69–74, Apr. 1995.

[7] "SimpleScalar Simulation Tools for Microprocessor and System Evaluation," *URL:http://www.simplescalar.org/*.