九州大学学術情報リポジトリ
Kyushu University Institutional Repository

タグ比較結果の再利用によるキャッシュメモリの低消費電力化

井上, 弘士
福岡大学工学部電子情報工学科

Moshnyaga G., Vasily
Department of Electronics and Computer Science, Fukuoka University

村上, 和彰
九州大学大学院システム情報科学研究院

https://hdl.handle.net/2324/3439

出版情報：電子情報通信学会技術研究報告, CPSY2001-61-74. 101 (473), pp.49-54, 2001-11. 電子情報通信学会CPSY研究会
バージョン：
権利関係：

タグ比較結果の再利用によるキャッシュメモリの低消費電力化

井上 弘士 † Moshnyaga G. Vasily † 村上 和彰 ‡

† 福岡大学 工学部 電子情報工学科

〒 814-0133 福岡県福岡市城南区七隈 8–19–1
E-mail: {*koji,vasily*}*@fukuoka-u.ac.jp*

‡ 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: *murakami@c.csce.kyushu-u.ac.jp*

あらまし：本稿では，低消費エネルギー化を実現する新しい命令キャッシュ・アーキテクチャとして，ヒストリ・ベース・ルックアップ・キャッシュ(HBL キャッシュ) 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．あるデータを格納可能なキャッシュ内ロケーションが複数存在するセット・アソシアティブ・キャッシュでは，参照データが唯一のウェイにのみ存在する（ヒットの場合）．それにも関わらず，従来型キャッシュでは，アクセス時間を短縮するために全てのウェイが並列に検索される．これに対し，HBL キャッシュは，過去のタグ比較結果を再利用し，参照データ検索における無駄なウェイ・アクセスを回避することで，低消費エネルギー化を実現する．ベンチマーク・プログラムを用いた定量的評価を行った結果，従来型キャッシュと比較して，約 0.2%の性能低下を伴うだけで，最大 72%のキャッシュ・アクセス消費エネルギーを削減できた．

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，再利用

A Low Power Cache Memory Architecture based on Tag Compare Reuse

Koji Inoue † Moshnyaga G. Vasily † Kazuaki Murakami ‡

† Department of Electronics and Computer Science, Fukuoka University

E-mail: {*koji,vasily*}*@fukuoka-u.ac.jp*

‡ Department of Informatics, Kushu University

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

Abstract : This paper proposes a novel architecture for low-power instruction caches called "*history-based look-up cache (HBL cache)*". In conventional n-way set-associative caches, there are n locations where a cache line can be placed in the cache space, and all ways are activated on every cache access because of the parallel search strategy. On the other hand, the HBL cache attempts to reuse the tag comparison results, and reduces the cache-access energy by avoiding the unnecessary way activations. The tag-comparison results are recorded in an extended BTB (Branch Target Buffer) for branch prediction. In our evaluation, it is observed that the HBL cache reduces the energy consumption by about 72 %, while it degrades the performance by only 0.2 %, compared with a conventional set-associative cache.

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

1　はじめに

現在のマイクロプロセッサ・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，を同時に達成できる．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．しかしながら，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，低消費エネルギー化を目的とした命令キャッシュ向けアーキテクチャとして，ヒストリ・ベース・ルックアップ・キャッシュ(HBL キャッシュ) 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．あるデータを格納可能なキャッシュ内ロケーションが複数存在するセット・アソシアティブ・キャッシュ(以下，SA キャッシュ) では，参照データが唯一のウェイにのみ存在する（ヒットの場合）．それにも関わらず，キャッシュ・アクセスが発生した際，従来方式では全てのウェイが検索対象となる．これに対し，HBL キャッシュは，過去のタグ比較結果を再利用して，検索対象となるウェイをキャッシュ・アクセス開始前に特定する．これにより，無駄なウェイ・アクセスを回避し，低消費エネルギー化を実現できる．タグ比較結果を記録するためのメモリ領域としては，多くの高性能プロセッサで搭載されている分岐予測バッファ (Branch Target Buffer:BTB) を用いる [1]．

以下，第 2 章では，従来型 SA キャッシュの動作とその消費エネルギーに関して説明する．次に，第 3 章では HBL キャッシュの詳細を示し，第 4 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 5 章で簡単にまとめる．

2　従来型セット・アソシアティブ・キャッシュ

CPU からのメモリ参照が発生した時，従来型の n ウェイ SA キャッシュ (n は連想度であり，ここでは 4 と仮定する) は次のように動作する．まず，プロセッサが出力したアドレスをデコードして，検索すべきキャッシュ内セットを決定する．次に，全ウェイ (ウェイ 0〜ウェイ 3) において，検索セットに対応するタグとラインをタグ・メモリおよびデータ・メモリから同時に読み出す．そして，読み出した 4 つのタグと，プロセッサが出力した参照アドレスのタグ・フィールド の値を比較する．一致するタグが存在 (ヒット) すれば，タグ比較結果に基づき，読み出した 4 つのラインの中から参照ラインを選択する．一方，一致するタグが存在しない (ミス) 場合，データ・メモリより読み出した 4 つのラインを全て破棄し，キャッシュ・リプレイスを行う．

このようなキャッシュ動作において，以下のエネルギーが消費される [6]．

$$E_{CACHE} = E_{dec} + E_{sram} + E_{io}$$

ここで，E_{dec} はデコード回路において消費されるエネルギーであり，参照アドレスの遷移確率に依存する．また，E_{io} は外部入出力ピン駆動に要するエネルギーであり，キャッシュ・ミス率に依存する．一方，E_{sram} は SRAM セルへのアクセスに要するエネルギーであり，以下の式に示すように，タグ・メモリおよびデータ・メモリにおいて消費されるエネルギー (E_{tag} と E_{line}) の和で表される．

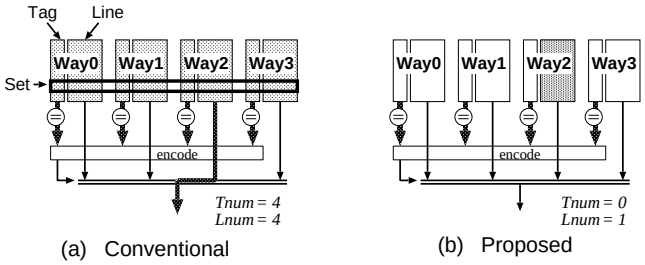

図 1: 提案手法による低消費エネルギー化

$$\begin{aligned} E_{sram} &= E_{tag} + E_{line} \\ &= Tnum \times E_{tag_acc} + Lnum \times E_{line_acc} \end{aligned}$$

ここで，E_{tag_acc} および E_{line_acc} は，タグ 1 個およびライン 1 個当りの読み出しに要するエネルギーを表す．また，$Tnum$ と $Lnum$ は，それぞれ，プログラム実行において読み出されるタグおよびラインの総数である．

3　HBL キャッシュ・アーキテクチャ

3.1　基本概念

従来型 SA キャッシュでは，唯一のウェイにのみ参照データが存在するにも関わらず，図 1(a) に示すように全てのタグ・メモリとデータ・メモリが活性化される (つまり，$Tnum = 4$，$Lnum = 4$)．もし，キャッシュ・アクセス前に参照命令が滞在しているウェイを特定することができれば，図 1(b) に示すように，活性化すべきメモリ・アレイ数を削減して低消費エネルギー化を実現できる（つまり，$Tnum = 0$，$Lnum = 1$ となる)．

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た時 (または，あるデータが追い出された時) 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．

このような事実に基づき，参照命令がキャッシュ中のどのウェイに滞在しているかをキャッシュ・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，ループ内に存在するある命令の実行について考える．最初にこの命令を参照する時，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（つまり，従来型キャッシュと同様に参照データの検索を行う必要がある)．しかしながら，次のループ・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

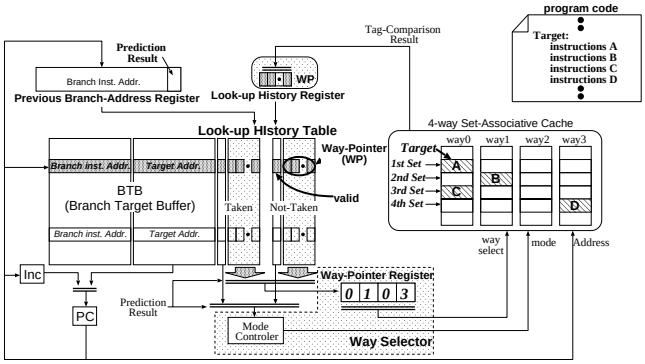

図 2: HBL キャッシュの内部構成 (連想度 4 の場合)

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)，この命令は前参照時と同じキャッシュ内ウェイに滞在していることが保証される．つまり，以下で説明するように，ある命令の参照に関して過去のデータ検索結果を再利用することで，命令キャッシュの低消費エネルギー化を実現できる．

1. 当該命令が最初に参照される際には，従来のキャッシュと同様にデータ検索を行う (図 1(a))．また，データ検索結果 (つまり,「どのウェイに当該命令が存在するか？」を示すタグ比較結果) を専用メモリ領域に記録する．

2. 当該命令の次参照において，前回の参照から現在に至るまでキャッシュ・ミスが発生していなければ，上記 1 にて記録したデータ検索結果を再利用する．これにより，キャッシュ・アクセス開始前に検索すべきウェイを特定でき，活性化するメモリ・アレイ数を削減できる (図 1(b))．

3. もし，前回の参照から現在までの間にキャッシュ・ミスが発生した場合，当該命令はキャッシュから追い出されている可能性がある．そこで，キャッシュ・ミスが発生した際には上記 1 にて記録したデータ検索結果を破棄し，以降の当該命令に関する参照を上記 1 と同様に行う．

3.2 内部構造

第 3.1 節で述べたように，SA キャッシュの低消費電力化を達成するためには，命令参照に関する過去のタグ比較結果 (つまり，当該命令が存在するウェイ番号) を専用メモリ領域に記録する必要がある．HBL キャッシュでは，この専用メモリ領域として，分岐予測機構における BTB(Branch Target Buffer) を利用する．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，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．

HBL キャッシュにおける BTB の構成を図 2 に示す．従来の BTB 各エントリにおいて,「分岐成立用」および「分岐不成立用」それぞれに関する以下のハードウェア機構を追加する．

- ウェイ・ポインタ *(WP)*：対応するキャッシュ・ラインが存在するウェイ番号を示すフラグ．各 BTB エントリには n 個の WP が実装される．複数個の WP を実装することで，連続する複数キャッシュ・ラインのウェイ情報を記録できる．分岐成立用 WP は，分岐先命令列 (対応するエントリ内の分岐先アドレスから始まる命令列) に関するウェイ情報を示す．一方，分岐不成立用 WP は，当該分岐以降の命令列 (対応する分岐命令の次アドレスから始まる命令列) に関するウェイ情報を示す．

- 有効 *(valid)* フラグ：同一エントリに格納された全ての WP が有効であるか否かを示すフラグ．

また，HBL キャッシュの動作を制御するために，以下のハードウェア機構が必要となる．

- ウェイ・ポインタ・レジスタ *(WP* レジスタ*)*：BTB から読み出された WP を記憶するレジスタ．

- ルックアップ・ヒストリ・レジスタ *(LH* レジスタ*)*：HBL キャッシュが通常動作する際，連続するキャッシュ・ライン・アクセスに関するタグ比較結果を一時記憶するためのレジスタ．

- 前分岐命令アドレス・レジスタ *(PBA* レジスタ*)*：前分岐命令のアドレスとその分岐予測結果を保持するためのレジスタ．

- 動作モード・コントローラ *(MC)*：次節で示すアルゴリズムに従って，HBL キャッシュの動作モードを決定する専用回路．

3.3 動作

HBL キャッシュは，以下に示す 3 つの動作モードを有する．

- 検索省略モード (OMitting-mode:OM モード)：WP レジスタに格納された各 WP の値 (過去に記録されたタグ比較結果) に基づき，キャッシュ・アクセス時のウェイ選択を行う．つまり，図 1(b) で示したように，参照命令を含むウェイのみを活性化する．

- 記録モード (Tracing-mode:T モード)：従来型キャッシュと同様 (図 1(a)) に，全てのウェイにおいてデータ検索を行う．また，連続アクセスされる各キャッシュ・ラインに関して，タグ比較結果 (当該ラインが存在するウェイ番号) を LH レジスタに記録する．

- 通常モード (Normal-mode:N モード)：従来型キャッシュと同様 (図 1(a)) に，全てのウェイにおいてデータ検索を行う．T モードとは異なり，タグ比較結果の記録は行われない．

つまり，HBL キャッシュでは，OM モード動作時にのみ消費エネルギーの削減を期待できる．HBL キャッシュにおける動作モードの状態遷移を図 3 に示す．HLB キャッシュは以下のように動作する．

1. プログラム実行において BTB アクセスがヒットした場合，当該ヒット・エントリから分岐予測結果

に対応した WP と有効フラグが読み出される．読み出された WP が無効であれば，BTB アクセスで使用した PC と分岐予測結果を PBA レジスタに格納し，動作モードは T モードへと遷移する (下記 2 へ)．一方，読み出された WP が有効であれば，それを WP レジスタに格納し，動作モードは OM モードへと遷移する (下記 3 へ)．

2. 動作モードは T モードである．したがって，従来型キャッシュと同様，タグ比較に基づくデータ検索を行う．また，タグ比較結果に基づき，連続するキャッシュ・ラインが存在するウェイ番号を LH レジスタに順次書き込む (キャッシュ・ライン境界は PC を監視する事で検出できる)．本モードにおいて BTB ヒットが発生した場合には，PBA レジスタの値をアドレスとして，BTB エントリに LH レジスタの内容を書き込む．また，対応する有効フラグをセットする (上記 1 へ)．もし，本モードにおいて，LH レジスタの最終 WP に書き込みを行い，その後，キャッシュ・ライン境界が検出された場合には，WP 書き込みオーバフローが発生し，HBL キャッシュは N モードとなる (下記 4 へ)．この場合，記録した LH レジスタの内容は破棄される．

3. 動作モードは OM モードである．WP レジスタに格納された各 WP は，これから連続して参照されるキャッシュ・ラインが存在するウェイ番号を示している．そこで，PC を監視する事でキャッシュ・ライン境界を検出し，キャッシュ・ライン境界が検出される度に順次 WP を選択する．これにより，HBL キャッシュは，図 1(b) に示すように参照命令が存在するウェイを直接選択できる．本モードにおいて，最終 WP が選択されており，かつ，キャッシュ・ライン境界が検出された場合には，WP 読み出しオーバフローが発生して HBL キャッシュは N モードとなる (下記 4 へ)．一方，BTB ヒットが発生した場合には，上記 1 に従って動作する．

4. 動作モードは N モードである．よって，従来型キャッシュと同様に，全てのウェイに対するデータ検索を行う．BTB ヒットが発生した場合，上記 1 に従って動作する．

なお，命令キャッシュ・ミスが発生した場合には，有効な WP に対応するキャッシュ・ラインがキャッシュ外に追い出される可能性がある．そのため，図 3 に示すように，キャッシュ・ミスが検出された時には動作モードを N モードとする．これと同時に，すべての有効フラグをリセット (つまり，記録した全てのタグ比較結果を消去) する．また，BTB エントリの追い出しが発生した場合には，有効な WP の数を判定できなくなる (T モード時，次の BTB ヒットが発生した時点で LH レジスタの内容を BTB に格納するため)．よって，この場合においても，命令キャッシュ・ミス発生時と同様に動作する．さらに，分岐予測ミスが検出された場合，ならびに，RAS(Return Address Stack) にヒットした場合には，現在の動作モードに関係なく N モードに状態を遷移する．ただし，これらの場合には，記録された WP の無効化は発生しない．

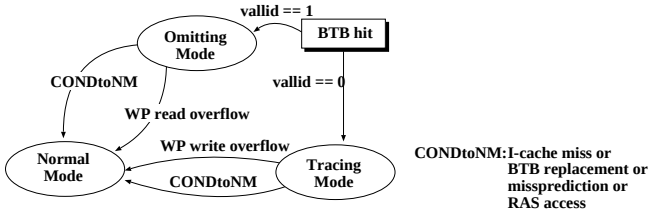

図 3: 動作モードに関する状態遷移

3.4 利点と欠点

HBL キャッシュにおける消費エネルギーは，以下の式で近似できる．

$$E_{TOTAL} = E_{CACHE} + E_{HBLoh}$$

ここで，E_{HBLoh} は，BTB 拡張によって生じる消費エネルギー・オーバヘッド (E_{btbadd})，ならびに，HBL キャッシュの動作モード 等を制御するための周辺論理回路で消費されるエネルギー (E_{logic}) の和で近似できる．

$$E_{HBLoh} = E_{btbadd} + E_{logic}$$

HBL キャッシュでは，OM モードで動作する場合，第 3.1 節の図 1(b) で示したように，参照命令が存在するウェイのみを活性化する．よって，プログラム実行において読み出されるタグ ($Tnum$) およびライン ($Lnum$) の総数は以下のようになる．

$$\begin{aligned} Tnum &= (TMnum + NMnum) \times W \\ Lnum &= (TMnum + NMnum) \times W + OMnum \times 1 \end{aligned}$$

ここで，W はキャッシュ連想度を表す．また，$OMnum$，$TMnum$，ならびに，$NMnum$ は，それぞれ，OM モード，T モード，N モードにおけるキャッシュ・アクセス回数である．以上より，HBL キャッシュでは，消費エネルギーに関して以下のような利点と欠点がある．

- OM モードで動作する場合には，無駄なウェイ・アクセスを回避するため，大幅な消費エネルギー削減効果を期待できる．
- 一方，T モードや N モードで動作する場合には，従来型キャッシュと同様に全てのウェイが活性化される．また，これに加え，BTB の拡張や周辺論理回路の追加による消費エネルギー・オーバヘッド ($Ebtb_add$) が生じる．

次に，HBL キャッシュの性能に関して議論する．キャッシュ性能は，ミス率とアクセス時間によって決定される．従来型と同じ構成 (同一キャッシュ・サイズ，ラインサイズ，連想度) の場合，HBL キャッシュにおけるミス率は，従来型キャッシュのそれと同じである．また，タグ比較結果を再利用するためのハードウェア機構はキャッシュのクリティカル・パス上に存在しないため，アクセス時間に関するオーバヘッドも発生しない．しかしながら，HBL キャッシュでは，BTB に対してタグ比較結果の読み出し / 書き込み要求が発生する．したがって，これらの BTB アクセス要求が受け付けられるている間，CPU からの BTB アクセスが禁止される．その結果，命令フェッチが停止

表 1: 4 ウェイ HBL キャッシュにおける消費エネルギー

Benchmark	Energy Consumption						Performance
	Cache Look-up Count	E_{tag} [uJ]	E_{data} [uJ]	E_{output} [uJ]	E_{btbadd} [uJ]	E_{TOTAL} [uJ]	Execution Time [clock cycle]
099.go	570,554,720 (0.739)	440,457	6,243,422	130,240	73,788	6,887,908 (0.825)	571,152,424 (1.013)
124.m88ksim	64,458,524 (0.447)	49,460	869,311	14,329	18,578	951,679 (0.617)	69,403,572 (1.027)
126.gcc	1,091,527,464 (0.648)	844,870	12,586,507	306,949	215,461	13,953,788 (0.764)	1,369,322,238 (1.022)
129.compress	11,171,513 (0.250)	8,555	208,555	3,899	4,972	225,983 (0.474)	20,798,875 (1.000)
130.li	53,252,741 (0.219)	40,783	1,082,823	21,241	35,332	1,180,180 (0.455)	114,407,367 (1.002)
132.ijpeg	814,158,977 (0.508)	623,528	10,328,448	141,044	91,927	11,184,949 (0.653)	698,540,345 (1.001)
102.swim	543,002,352 (0.621)	415,814	6,313,305	76,250	44,126	6,849,497 (0.733)	661,788,288 (1.000)
adpcm_enc	1,625,374 (0.171)	1,244	39,199	828	1,794	43,067 (0.425)	4,576,912 (1.003)
adpcm_dec	807,394 (0.108)	618	27,591	651	1,442	30,304 (0.380)	3,789,218 (1.003)
mpeg2_enc	344,687,046 (0.194)	263,955	7,598,909	154,573	189,887	8,207,325 (0.434)	803,730,640 (1.002)
mpeg2_dec	24,614,697 (0.126)	18,859	747,699	17,265	14,666	798,490 (0.381)	98,034,091 (1.002)

し，ストール・サイクルが発生する．具体的には，HBL キャッシュが以下の処理を行う間，CPU は BTB アクセスを待たなければならない．

- BTB に格納された全 WP の無効化 (全ての有効フラグをリセット)．本処理は，命令キャッシュ・ミス，または，BTB エントリの追い出しが発生した際に行われる．本処理に起因するストール・サイクル数は，BTB の実装方式に依存する．
- LH レジスタの値 (記録された過去のタグ比較結果) の BTB に対する書き込み．これは，T モード動作中に BTB ヒットが発生した際に行われる．通常，BTB へのデータ書き込みは 1 クロック・サイクルで終了するため，本処理に起因するストール・サイクル数は 1 となる．

4 評価

4.1 評価環境

HBL キャッシュの有効性を明らかにするため，ベンチマーク・プログラムを用いた定量的評価を行った．以下，使用したベンチマーク・プログラムを示す．

- SPECint95 [10]: *099.go*, *124.m88ksim*, *126.gcc*, *129.compress*, *130.li*, *132.ijpeg* (using train input).
- SPECfp95: *102.swim* (using test input).
- Mediabench [9]: *adpcm_encode*, *adpcm_decode*, *mpeg2_encode*, *mpeg2_decode*

具体的には，SimpleScalar シミュレータ [8] を改良し，上記のプログラムを実行した．ここで，特に断りのない限り，命令キャッシュ・サイズは 16K バイト，ラインサイズは 32 バイト，キャッシュ連想度は 4，分岐予測テーブルの連想度は 1，エントリ数は 2048，分岐予測器は 2 ビット bimod モデル，BTB の連想度は 4，セット数は 512 と仮定する．また，その他のパラメータに関しては，SimpleScalar シミュレータのデフォルト値を用いた．また，命令フェッチ後にプリ･デコードを行う事により，BTB にアクセスすべき命令か否かを BTB アクセス前に判別可能と仮定する (プリ・デコードを行わない場合については，第 4.4 節にて議論する)．HBL キャッシュに関して，BTB エントリ当たりの WP 数は分岐成立用/不成立用と共に 4 とする．また，WP 無効化処理に要する遅延時間は 1 クロック・サイクルと仮定する．

一方，HBL キャッシュの消費エネルギーは，第 2 節ならびに第 3.4 節より，以下の式で表される．

$$\begin{aligned} E_{TOTAL} &= E_{CACHE} + E_{HBLoh} \\ &= E_{dec} + E_{tag} + E_{line} + E_{io} + E_{btbadd} + E_{logic} \end{aligned}$$

ここで，アドレス･デコードにおける消費エネルギー E_{dec} は，E_{sram} および E_{io} に比べ，E_{CACHE} に与える影響が極めて小さいことが報告されている [2]．また，HBL キャッシュの制御回路は単純な状態遷移機械で実現できる．そこで本評価では，E_{dec} ならびに E_{logic} を 0 と仮定する．その結果，HBL キャッシュの消費エネルギーは以下のようになる．

$$E_{TOTAL} = E_{tag} + E_{line} + E_{io} + E_{btbadd}$$

これらの各消費エネルギー要素に関しては，文献 [3] で提案されたキャッシュ消費エネルギー・モデルを参考にして求めた．その際，0.8um CMOS テクノロジを想定し，文献 [4][7] で示された各種パラメータ (負荷容量) を参照した．なお，従来型キャッシュでは，上記の消費エネルギー式において E_{btbadd} が 0 となる．

4.2 実験結果

表 1 に実験結果を示す．ここで，表中の ( ) 内の数字は，従来型 4 ウェイ SA キャッシュでのシミュレーション結果に正規化した値である．

半分以上のプログラムに関して，HBL キャッシュは，命令キャッシュ・アクセスにおけるデータ検索回数を 70%〜90%削減している (*129.compress*, *130.li*, *adpcm*, *mpeg2*)．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．また，*099.go* を除くその他全てのプログラムに関しても，35%〜50%の削減率である．この結果，表 1 で示すように，多くのプログラムにおいて 50%〜60%の消費エネルギー削減を達成した．また，その際の性能低下は 1%〜3%程度であった．

4.3 WP 無効化操作が性能に与える影響

第 3.4 節で示したように，WP キャッシュでは，命令キャッシュ・ミスが発生した場合，または，BTB エントリの追い出しが発生した場合に WP の無効化処理が行われる．第 4.2 節における評価では，WP 無効化により発生する CPU ストール・サイクル数は 1 と仮定した．そこで本節では，WP 無効化遅延時間が CPU 性能に与える影響を評価する．

4 つのベンチマークを対象とし，WP 無効化遅延時間 (クロック・サイクル数) を変化させた場合のプログラム実行時間を図 4 に示す．全ての結果は，従来型キャッシュ

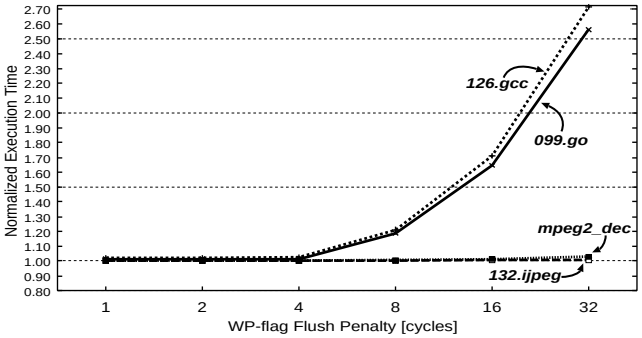

図 4: WP 無効化ペナルティが性能に与える影響

を用いた場合の実行時間に正規化している．無効化遅延時間が 8 サイクル未満の場合には，大幅な性能低下が発生していない．これは，以下のような理由による．

WP 無効化発生原因の内訳を解析した結果，95%以上が命令キャッシュ・ミスに起因していることが分かった (残り 5% が BTB エントリの追い出しに起因する)．これは，WP 無効化による殆んどの CPU ストールは命令キャッシュ・ミスが原因であることを意味する．命令キャッシュ・ミスが発生した場合，WP の無効化処理は命令キャッシュ・リプレイスメントと並列に行われる．つまり，WP 無効化遅延時間がキャッシュ・ミス・ペナルティより小さい場合，WP 無効化による性能オーバヘッドは隠蔽される．実際，本シミュレーションにおいては，命令キャッシュ・ミス時のミス・ペナルティを 6 クロック・サイクルと仮定した．

4.4 WP 数が消費エネルギーに与える影響

HBL キャッシュの各 BTB エントリには，分岐成立用/不成立用それぞれに複数個の WP が格納される．実装される WP 数が多い場合，より長い連続した命令列に関するウェイ情報を記録できる．その反面，BTB のビットライン数が増加するため，BTB アクセスにおける消費エネルギー・オーバヘッド (E_{btbadd}) が無視できなくなる．そこで，本節では，BTB エントリに格納される WP の数が消費エネルギー削減効果に与える影響を評価する．

第 4.2 節ならびに第 4.3 節では，プレ・デコーディングにより，分岐命令 (またはジャンプ命令) 実行時にのみ BTB アクセスが発生すると仮定した．各 BTB エントリにおいて，分岐成立用/不成立用 WP の数を変化させた場合の消費エネルギーを図 5 に示す．ここで，図 5(A) はプレ・デコーディングを行った場合であり，図 5(B) は命令フェッチ毎に BTB アクセスが発生する場合である．

プレ・デコーディングを行った場合 (図 5(A))，*132.ijpeg* を除く全てのプログラムに関して，命令キャッシュ内データ検索回数の削減率は WP の増加と共に飽和している．これは，連続実行される命令列の平均長が短いためと考える．その結果，WP 数の増加と共に E_{btbadd} が増加し，全消費エネルギーも増加傾向となる．これに対し，*132.ijpeg* に関しては，連続命令列の平均長が長いため，WP の増加と共に消費エネルギー削減効果も向上している．一方，命令フェッチ毎に BTB アクセスが発生する場合 (図 5(B))，WP 数の増加に伴う E_{btbadd} の増加が顕著に現れる．しかしながら，プレ・デコーディング実装の有無には関係

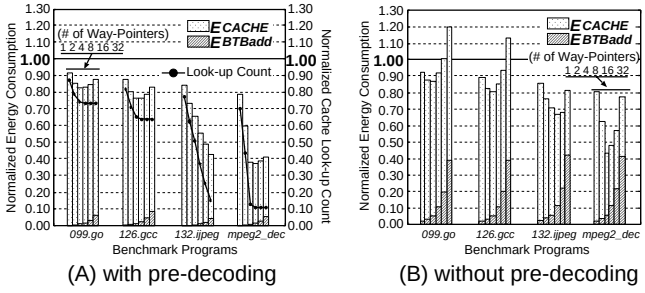

図 5: WP 数が消費エネルギーに与える影響

なく，WP 数を 4 程度にすることで，提案手法は従来型キャッシュから大幅な消費エネルギーの削減を達成している (*mpeg2_dec* では 50%)．

5 おわりに

本稿では，低消費電力キャッシュ・アーキテクチャとして，ヒストリ・ベース・ルックアップ・キャッシュ(HBL キャッシュ) を提案した．本方式では，プログラム実行中，タグ比較結果を拡張 BTB に記録する．そして，これを再利用することにより，命令キャッシュ内におけるデータ検索処理を省略し，低消費エネルギー化を実現する．

複数ベンチマークを用いて実験を行った結果，従来型キャッシュと比較して，最大 72%の消費エネルギーを削減できた．また，この時の性能低下は 0.2%であった．今後，実設計に基づく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 (課題番号：09358005, 11308011, 12358002, 13308015) による．

参考文献

[1] 井上弘士, 村上和彰, “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,” 報処理学会研究報告, 2000-ARC-140, pp. 25–30, 2000 年 11 月.

[2] 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.

[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 the 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] S. J. E. Wilton and N. P. Jouppi, “An Enhanced Access and Cycle Time Model for On-Chip Caches,” *WRL Research Report 93/5*, July 1994.

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

[9] MediaBench, *URL: http://www.cs.ucla.edu/l̃eec/mediabench/*.

[10] SPEC (Standard Performance Evaluation Corporation), *URL:http://www.specbench.org/osg/cpu95*.