タグ比較結果の再利用に基づくメディア・アプリケーション向け低消費電力キャッシュ

井上, 弘士
福岡大学工学部電子情報工学科

Moshnyaga, Vasily G.
Dept. of Electronics Engineering 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

出版情報：第15回 回路とシステム（軽井沢）ワークショップ, pp.571-576, 2002-04. 電子情報通信学会
バージョン：
権利関係：

タグ比較結果の再利用に基づく メディア・アプリケーション向け低消費電力キャッシュ

A Low Power Cache Architecture based on Tag-Comparison Reuse for Media Applications

井上弘士 † モシニャガ・ワシリー † 村上和彰 ††
† 福岡大学 工学部 電子情報工学科
†† 九州大学大学院 システム情報科学研究院

Koji INOUE† Vasily MOSHNYAGA† Kazuaki MURAKAMI††
†Dept. of Electronics Engineering and Computer Science, Fukuoka University
††Dept. of Informatics, Kyushu University

1 はじめに

増大を続けるメモリ・アクセス・レイテンシの隠蔽を目的として，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．

本稿では，新しい低消費エネルギー・キャッシュとして，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．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，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することで，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．本手法は命令の繰返し実行性を積極的に活用するため，固定的なループ構造を多く含むメディア・アプリケーションに対して特に有効である．

文献 [5] にて，ライン構成を拡張することでタグ比較結果を再利用する手法が提案された．これに対し，本稿で提案する技術は分岐予測バッファを活用しており，実装方法は全く異なる．CPU が分岐予測ユニットを搭載している場合，ハードウェア・コストを小さくすることができる．また，文献 [1] では，本手法の提案および基本評価を行った．本稿では，これらの評価結果に加え，他技術との定量的比較も行う．

以下，第 2 章では，従来型 SA キャッシュの動作とその問題点を明らかにする．次に，第 3 章では HBL キャッシュの詳細を示し，第 4 章では定量的評価を行う．最後に，第 5 章で簡単にまとめる．

2 従来型 SA 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．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．

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

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．

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

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

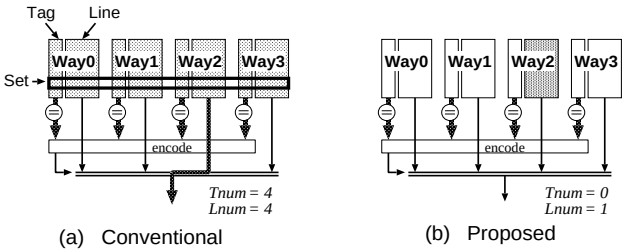


図 1: 提案手法による低消費エネルギー化

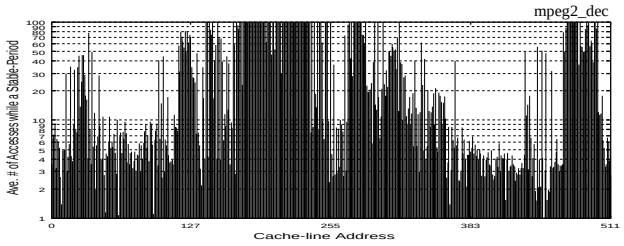


図 2: キャッシュ安定期間における平均ライン・アクセス回数

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．

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，アクセス当たりの $Tnum$ および $Lnum$ は共に 4)．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．1) 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．2) 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．したがって，あるキャッシュ・ミスが発生し，次にキャッシュ・ミスが発生するまでの間 (このような期間を**キャッシュ安定期間**と呼ぶ)，キャッシュの内容は全く変化しない．

キャッシュ安定期間において，ある命令 iはキャッシュ内の同一ロケーションに滞在する．よって，キャッシュ安定期間に命令 iが N 回参照される場合，全当該アクセスにおける検索結果 (タグ比較結果) は同じとなる．つまり，本質的に実行すべきデータ検索処理は最初のアクセス時のみであり，残り N-1 回のアクセスにおけるデータ検索処理は冗長となる．キャッシュ安定期間当たりの平均キャッシュ・ライン・アクセス回数 (最大 100 回まで) を図 2 示す．図 2 において，横軸はキャッシュ・ライン・アドレスを表している (実験環境の詳細に関しては第 4.1 を参照)．キャッシュ安定期間において，いくつかのキャッシュ・ラインは平均 100 回以上参照されている．しかしながら，従来型キャッシュはアクセス毎にデータ検索処理を行うため，多くの無駄なエネルギーを消費する．

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，図 1(a) に示すよう，アクセス毎に全てのタグ・メモリとデータ・メモリが活性化される．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，アクセス当たりの $Tnum$ および $Lnum$ は，それぞれ，0 と 1)．

HBL キャッシュは，過去のタグ比較結果を再利用する．これにより，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) 判定し，無駄なキャッシュ検索による消費エネルギーを削除する．つまり，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の検索を行う (図 1(a))．これと同時に，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．そして，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，記録した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))．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，これまでに記録した全ての検索結果を破棄 (無効化) する．

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

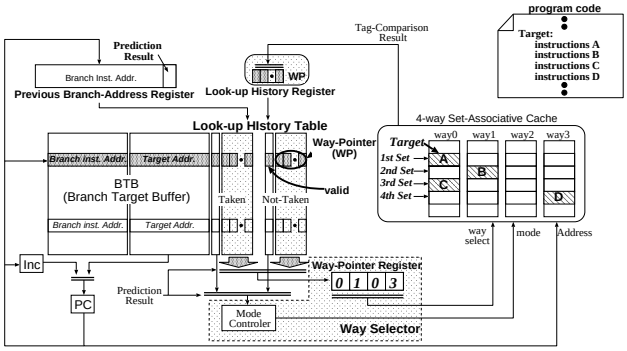


図 3: HBL キャッシュの内部構成 (連想度 4 の場合)

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 3 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, 次に示す 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 4 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 レジスタに格納して 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). 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する. この場合, これまで LH レジスタに記録したタグ比較結果を BTB エントリに書込み (PBA レジスタの値を BTB アクセス・アドレスとして使用), N モードへと状態を遷移する (下記 4 へ).

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

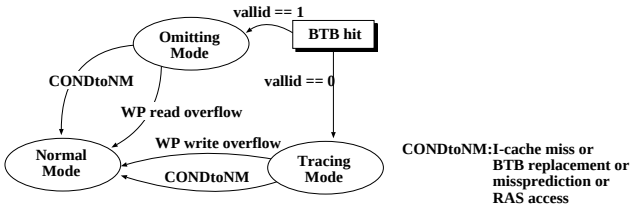


図 4: 動作モードに関する状態遷移

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 4 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．

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．

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

よって，HBL キャッシュが OM モードで動作する場合には，図 1(b) に示すように，無駄なウェイ・アクセスを回避することで大幅な消費エネルギーを削減できる．しかしながら，T モードや N モードで動作する場合には，図 1(a) に示すように，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．ここで，タグ比較結果の読出しは，図 3 に示すように，分岐先アドレス読出しと並列に行うことができる (アドレス・デコード結果は共有できるため)．しかしながら，タグ比較結果の書込み，または，無効化処理においては，分岐先アドレス読出しのための BTB アクセスとアドレス・デコーダを共有できない．したがって，タグ比較結果の更新処理が行われている間，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し，ストール・サイクルが発生する．

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

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)

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 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．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．

HBL キャッシュの消費エネルギーに関しては，文献 [3] を参考にして以下の式で近似した．

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

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．

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

なお，本評価では，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．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

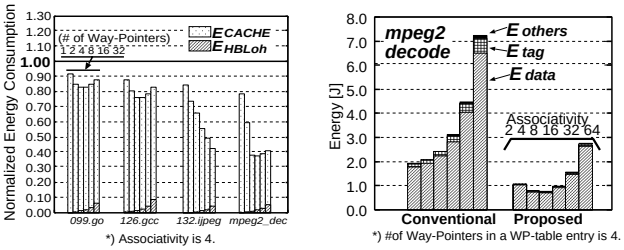
表 1 に実験結果を示す．ここで，表中の ( ) 内の数字は，従来型 4 ウェイ SA キャッシュでのシミュレーション結果に正規化した値である．

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．この性能低下は，第 3.4 で述べたように，BTB に対する WP 書込みアクセスと，分岐予測のための分岐先アドレス読出しアクセスの競合にって生じる．したがって，BTB において WP 書込み専用ポートを設けることで，この性能低下を隠蔽することができる．

4.3 他パラメータが与える影響

図 5(A) に，エントリ当たりに格納される WP 数を変化させた場合のキャッシュ消費エネルギー (E_{TOTAL}) を示す．全ての結果は従来型キャッシュにおける消費エネルギーで正規化している．WP 数を増加することで，より多くのタグ比較結果を再利用できる．その結果，キャッシュ・メモリでの消費エネルギー (E_{CACHE}) が削減する．しかしながら，BTB アクセスにおける消費エネルギー・オーバヘッド (E_{HBLoh}) も増加するため，WP 数が 4(または 8) 以上になった場合，逆に消費エネルギーが増大する．

一方，図 5(B) に，キャッシュ連想度を変化させた場合 (WP 数は 4 で固定) の E_{TOTAL} を示す．従来型キャッシュでは，連想度の増加と共に消費エネルギーも増加する．これに対し，HBL キャッシュでは，連想度が 4〜8 に増加した場合でも消費エネルギーは削減している．これは，直接ウェイ選択による低消費エネルギー効果が顕著に現れたためである．しかしながら，連想度が 8 以上となった場合，消費エネルギーは従来型キャッシュと同様に増加傾向となる．これは，直接ウェイ選択による低消費エネルギー効果よりも，連想度の極端な増加に伴うオーバヘッドが上回ったためである．



(A) Effects of number of WPs (B) Effects of associativity

図 5: その他のパラメータが与える影響

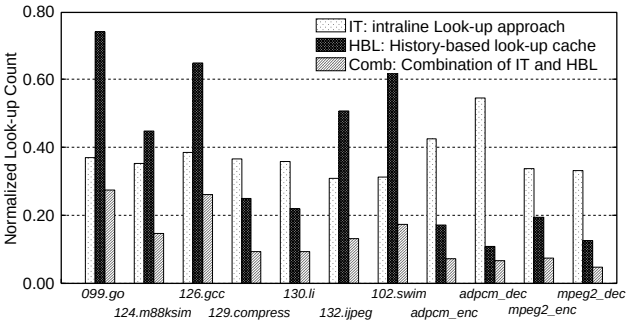


図 6: IT アプローチとの比較

4.4 他技術との比較

通常，データ検索は，複数データ (連続する複数命令)が格納されるキャッシュ・ライン単位で行われる．よって，命令 i と j が連続して実行される時，これらの命令が共に同一キャッシュ・ラインに存在する場合，命令 j の参照におけるデータ検索処理 (タグ比較処理) を省略できる [6]．このような手法を本稿では IT(Intraline Tag-check) と呼ぶ．IT キャッシュ，HBT キャッシュ，ならびに，これらの組み合わせ (Comb) に関して，プログラム実行時における総データ検索回数を評価した．その結果を図 6 に示す．

各プログラムにおいて，全ての結果は従来型キャッシュの結果に正規化している．一般に，分岐を除く全てのプログラム・コードにおいて，各命令は逐次実行 (インクリメンタル実行) される．そのため，基本的に連続する命令は同一キャッシュ・ラインに格納される確率が高い．よって，IT キャッシュは全てのプログラムにおいてデータ検索回数を 65〜70%削減している．これに対し，HBL キャッシュ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が様々である．例えば，mpeg や compress 等のメディア・アプリケーションに関しては 75〜90%データ検索処理を削減している．これらのプログラムには，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，IT キャッシュと HBL キャッシュを組合わせることで，より高いデータ検索削減効果を得ることができた．

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 62%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, M. Vasily，村上和彰, “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,” 信学技報, CPSY2001-69, pp. 49–54, 2001 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] A. Ma, M. Zhang, and K. Asanović,“Way Memorization to Reduce Fetch Energy in Instruction Caches,” *ISCA Workshop on Complexity Effective Design*, July 2001.

[6] 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.

[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*.