組込みプロセッサ向け命令キャッシュ制御方式の検討

請園智玲[†]　田中清史[†]

組込みアプリケーションの大規模化にともない，プログラムコードをオンチップメモリに収めることが困難となってきている．オフチップメモリを使用することは，プロセッサ内の命令キャッシュのヒット率が性能に大きく影響してくることを意味する．本論文では従来のキャッシュメモリ制御の問題点を整理し，キャッシュミス時のフィル動作を制御することにより，参照局所性の乏しいブロックがキャッシュメモリに挿入されることによるヒット率低下を防ぐ方式を提案する．また，提案方式をスラッシング回避に適用する方法を示し，この方法を MiBench を使用してシミュレーションした結果，平均約 20%のミス率削減を達成した．

A Study of Control of Instruction Caches in Embedded Processors

Tomoaki Ukezono[†] and Kiyofumi Tanaka[†]

As embedded applications grow, it's difficult for on-chip memories to accommodate the whole program codes. The use of off-chip memories means that miss-hit ratio of instruction caches in embedded processors affects the total performance. To improve the hit ratio, this paper discloses problems of conventional cache memories and proposes a technique that prevents blocks with little locality from being inserted to instruction caches. In addition, we describe the application of the proposed cache system to thrashing. Simulation results using MiBench show effectiveness of the proposed technique, reduction of misses by 20% on average.

1. はじめに

近年の組込みシステム開発では開発効率とリアルタイム性の確保の観点から組込み／リアルタイム OS を利用することが一般的になりつつあり，バイナリサイズは増大傾向にある．組込みプロセッサは高性能汎用プロセッサと比較し一次（L1）キャッシュ容量が小さく，二次（L2）キャッシュを持たないものが一般的であり，L1 ミスペナルティを L2 で緩和できないことから，アプリケーションの規模が大きくなると命令キャッシュミスが増加し大きな性能低下を引き起こす．

キャッシュヒット率向上のための方策としてコードサイズを削減することが挙げられる．その一つに，異なる命令長（16 ビット，32 ビット）が混在する RISC 命令セットアーキテクチャ[1),2)] の使用がある．これは 32 ビット命令による実行の高効率と 16 ビット命令によるコード密度効率を両立することが狙いであるが，16 ビット命令を使用した部分の実行効率が低くなることは避けられない．[3)]

その他のコードサイズ削減方針として圧縮技術の利用がある．組込み向けコード圧縮方式として様々な研究が存在する．[4)5)] 高圧縮率のアルゴリズムの使用によりコードサイズ削減が得られる反面，実行時に解凍処理のオーバヘッドが存在するため，参照頻度の高いキャッシュ内の命令やデータには適用困難である．

キャッシュミスを低減するために，ARM ファミリの中にはプログラムから制御可能な高速なローカル密結合メモリ（TCM: tightly coupled memories）を持つものがある．同様に Cell BE の SPE が持つローカルストア (LS)[6)] はプログラム制御可能な高速メモリである．このようなスクラッチパッドメモリはプログラマによる明示的なプリロードを前提とするため，通常のキャッシュと比較し，キャッシュミスという予測困難なレイテンシ増大は防ぐことができるものの，プログラマの負担が存在する．

また，高連想度キャッシュを提供することによりキャッシュミス削減を狙うプロセッサが存在する．(StrongARM[7)] (32-way)，XScale[8)] (32-way) など.) 高連想度キャッシュでは高速タグ検索を可能とするために CAM を使用することになり，ハードウェアサイズおよび電力消費の観点からは低コストプロセッサでは採用困難である．[9)]

本研究では，低連想度の命令キャッシュをターゲッ

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

トとしてヒット率を向上させる方式を提案する．キャッシュサイズが限られているため，ある程度大きなアプリケーションでは，性能向上のために使用頻度の高いコード部分を命令キャッシュ内に意図的に留めることが有効であると予想した．

本稿は以下の構成を採る．2 節では従来の命令キャッシュの振舞いについて整理し，提案する方式の狙いを述べる．3 節では，キャッシュミスを削減する命令キャッシュのアーキテクチャを提案する．4 節で評価環境および評価結果を示し，最後に 5 節でまとめと今後の課題について述べる．

2. 従来の命令キャッシュの振舞いと改善可能性

従来の命令キャッシュではミスが発生すると，その命令を含むブロックは無条件にキャッシュにフィルされる．当該セットが他のブロックで満たされていた場合は LRU などのアルゴリズムに従って置換を行なう．LRU は各ブロックへの過去の参照時間を反映する情報を利用して置換ブロックを選択することから，メモリ参照パターンが繰り返される傾向があるプログラム実行に対しては最適な方法の一つである．

命令列の多くは時間的局所性を持つが，命令がプログラムコードのどの部分に属しているかにより，“参照間隔（access interval）” が異なる．キャッシュ内でヒットすると当該命令ブロックは当該セット内で最新となる．すなわち，参照間隔の短いブロックは頻繁に最新となる．したがって，再利用性があるブロック同士では，LRU は参照間隔の短いものをキャッシュ内に残す傾向がある．キャッシュ内に存在しているブロック同士に関してはこの方針は妥当であるが，ミスしたブロックは参照間隔に関わらず強制的にキャッシュへフィルされるため，より参照間隔の短いブロックを追い出す可能性があることが問題として挙げられる．

一方，ミスしたブロックをキャッシュへ挿入することは空間的局所性の観点からは重要である．さもなければ，同一ブロック内の連続する命令をフェッチする毎にミスが発生し大きなペナルティを被ることになる．しかしながら，空間的局所性は必ずしもキャッシュで対処する必要はなく，キャッシュと並列に参照可能なバッファを用意し，これにミスブロックを挿入することにより解決可能である．命令フェッチはデータ参照と異なり，(マルチスレッドアーキテクチャでない限りは）一系統の流れであるため，バッファはキャッシュブロックと同サイズのものが一つ存在すれば十分である．

本論文では命令キャッシュミス時のフィル制御方式を提案する．基本的な方針として，時間的局所性の低いブロックへの参照でミスが発生した場合はキャッシュへフィルせずに前述のバッファに挿入することにより，時間的局所性のより高いブロックに対するミスを軽減する．これにより，従来のキャッシュよりも時間的局所性をより反映することが可能であり，更に同一セット内で参照間隔の短いブロックの数が連想度（ウェイ数）を超える場合に発生するスラッシングを緩和できる．

命令ブロックの時間的局所性はブロック毎に異なるが，プログラムの制御構造と関連があることが期待できる．すなわち，フィル制御時の指標として以下のような実行単位との関連付けが考えられる．

(1) 手続き／関数単位
手続きによって呼び出される頻度は異なる．高頻度で呼び出される手続き実行時のキャッシュミスはフィルを行い，その他はバッファに挿入する．手続き単位を拡張し，アプリケーション内のタスク単位の制御に発展可能である．

(2) ループ内外
ループ内の実行は高い時間的局所性が期待できるため，ミス時にキャッシュにフィルする．ループ外実行でミスした場合はバッファに挿入する．

(3) 分岐／基本ブロック単位
実行頻度の高いパス上の基本ブロックに属するブロックはキャッシュフィル対象とし，それ以外はバッファ挿入対象とする．

(4) 命令単位
ミスした命令の単位で，その命令を含むブロックをフィルするべきかどうかを判断する．最も粒度の小さい制御単位であり，これを実現することは (1)〜(3) の全ての制御単位を含むことになる．

実行単位として最も粒度の小さいものは (4) であるが，実際のキャッシュハードウェアは数命令から成るブロック単位で管理されることから，次節ではブロック単位の選択方式を提案する．

3. 提案制御方式

本節では命令キャッシュフィル制御の実現方式を提案する．実際のミス時のフィル制御は，何らかの静的あるいは動的情報が与えられれば簡単なセレクタ機構となる．セレクタ機構をブラックボックス化した命令キャッシュシステムの構成を**図 1** に示す．図中の FTS（Fill Target Selector）がセレクタ機構である．

提案システムでは，命令フェッチ時に命令キャッシュと 1 ブロックエントリの命令バッファを並列にアクセスし，いずれかでヒットすれば命令を実行ユニットに

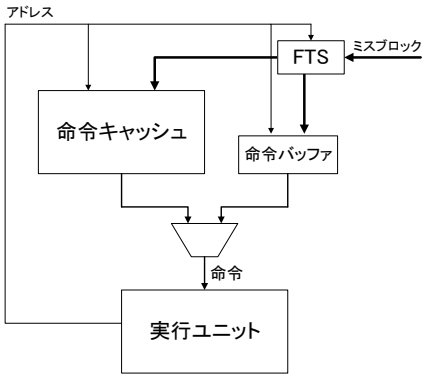

図 1 命令キャッシュシステムのブロック図.
Fig. 1 Block diagram of instruction cace system.

供給する．ミス時には低階層のメモリからミスブロックを読み出す．このミスブロックは後述する選択情報を利用して，FTS により命令キャッシュにフィルするか命令バッファに挿入するか判断される．

ブロック単位の選択方式の一つとして，1 ビットのフィル選択情報を各ブロックアドレスに関連付ける方法が挙げられる．全ブロックのフィル情報をメモリ内に用意し，キャッシュミス時にミスブロックと同時に読み出し FTS に与える．この方法ではフィル情報がメモリに格納されることになるが，メモリ使用オーバヘッドは命令サイズが 32 ビットでブロックサイズが 16 バイトのときに 0.8%となる．

もう一つの方法として，キャッシュのフィル対象，または非フィル対象のブロックアドレス群を登録するルックアップテーブルを FTS 内に用意することが挙げられる．フィル対象，あるいは非フィル対象のブロック数が比較的少ない場合は，上記の全ブロックに情報を用意する方法よりもハードウェア／メモリオーバヘッドが小さくなる．また，FTS による選択処理はキャッシュミスペナルティの間に行われるため，ルックアップテーブル参照は極めて高速である必要はない．本稿の評価ではこの方式を採用する．

フィル選択情報は，事前の試験的シミュレーションによるキャッシュミスに関するプロファイル等を参考にしてアプリケーション開発者あるいは開発環境が生成する．このことは開発工程の増加を意味するが，アプリケーション特化である組込みシステム開発においては現実的といえる．

4. 評　　価

以降の節では，前節で紹介したブロック単位でフィル先を選択する提案手法に関して，実現性を考慮せずに，シミュレーションベースで検証し，予備評価を行うと共に，提案手法の潜在的性能改善能力を議論する．

4.1 シミュレーション環境

本節では，提案手法を評価する環境を述べる．提案手法の性能を SimpleScalar4.0 コードベースの CPU シミュレータ SimpleSclar/ARM[10] で評価した．実行は最も精度の高いシミュレーション結果を得ることができる sim-outorder で実行した．本評価は命令キャッシュの評価であるため，いくつかの命令キャッシュ構成で提案手法を検証した．それ以外のシミュレーションパラメータは SimpleScalar のデフォルト値を使用している．命令キャッシュ構成は全てのシミュレーションにおいて，ブロックサイズを 16B とした．評価した命令キャッシュ構成は 512B-2Way，1KB-2Way，1KB-4Way，2KB-2Way，2KB-4Way の 5 種類の構成である．更に，個々のキャッシュ構成には命令バッファと FTS をシミュレーションモデルとして実装した．本評価の FTS はフィルするブロックのブロックアドレスに応じて，フィル先を選択するハードウェアであり，あらかじめ，“命令キャッシュに入れない” ブロックアドレスが格納されているものとした．FTS は無限に大きいルックアップテーブルを持ち，遅延無しで読みだせるものであると仮定したため，FTS の回路は実行に対して如何なるオーバヘッドも与えない．

ベンチマークは MiBench[11] の HP[12] より，MiBench Version 1.0 の ARM 用プリコンパイルバイナリを取得し使用した．本評価では，512B-2Way，ブロックサイズ 16B の命令キャッシュの構成を最小構成とし，その構成において，ミス率が 1%未満のアプリケーションは評価から除外した．評価対象となったのは，automotive から 3，consumer から 4，network から 2，office から 4，security から 2，telecom から 3 の計 18 アプリケーションである．全てのアプリケーションの入力データは large を選択した．

4.2 提案手法の評価方法

本稿では，キャッシュブロック単位のキャッシュミスしたブロックアドレスのトレースを事前実行で取得し，トレースを基にプロファイラがプロファイル（FTS にセットするブロックアドレス列）を作成し，それを FTS にセットすることで，提案手法の効果を計測する．

トレース収集実行で，命令キャッシュミスしたブロックアドレスのトレースを生成できるよう SimpleSclar/ARM を修正した．収集したトレースから，ブロックアドレス毎の総ミス数が集計される．集計したブロック毎のミス数の一例を図 2 に示す．

図は MiBench のアプリケーション，basicmath の実行で，512B-2Way で命令キャッシュを構成した時

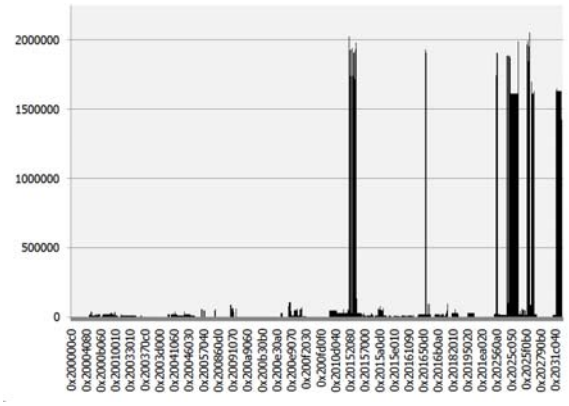

図 2 basicmath におけるブロックアドレス毎のミス数.

の各ブロックアドレスに対するミス数を示している．縦軸はミス数で，横軸は各ブロックアドレスを示している．収集できるミスブロックアドレスに連続性は保証されないので，横軸はアドレスで連続していない．図ではブロックアドレスを昇順でソートして表示している．図の大きな傾向として，ほとんどのブロックは 100〜2000 以内のミス数に収まっているのに対し，一部のブロックのみが 1,500,000 回以上のミスを発生させていることがわかる．このスパイクは本評価で対象とする 5 つの命令キャッシュ構成全てで観測された．本研究ではこのスパイクを同一セットに対する競合性ミスが原因のスラッシングであると推測する．プロファイラはこのスパイクに着目し，スラッシング緩和のために FTS を使用する．プロファイラは以下の手順で，命令キャッシュにフィルせずにバッファにフィルするブロックを抽出する．

(1) トレースからブロックアドレス毎のミス数を集計.
(2) 計測時のブロックサイズ，セット数から，セット毎のブロックアドレス（(1) で集計したミス数の情報を含む）の集合を生成.
(3) 生成した集合毎に最大ミス数を見つける.
(4) 最大ミス数の 1/2 以上のミスを発生させるブロックアドレスをスラッシングによるミスが多発するブロックアドレスとしてマークし，そのリスト生成.
(5) 生成したリストをミス回数で降順ソートし，上位から計測時の Way 数分を除外

以上の 5 手順を経て，作成されたリスト（セット数分存在）がバッファ挿入対象ブロックアドレスとなる．作成されたリストは統合されプロファイルとして，実行に使用される．このアルゴリズムはセット毎に競合するブロックで，スラッシングを発生させる可能性のあるブロックを抽出し，その中でも最も多くミスするブロックを Way に残留させ続け，残りのブロックをバッファに挿入することで，少なくとも Way 数分のブロックの潜在キャッシュヒットを確保することを目的としている．例えば，2Way 構成のキャッシュである特定セットに 3 ブロックが競合し，スラッシングを起こしており，結果的に各ブロックへの参照が 10,000 回のミスを発生させていたとするならば，そのままでは 30,000 回のミスを発生させることになる．本アルゴリズムはその 30,000 回のミスを 10,000 回に減らす目的を持つ．

4.3 評価結果

本節では 4.2 節で示した評価方法により，提案手法のキャッシュミス率に関する評価を行う．

4.3.1 各命令キャッシュ構成における効果

512B-2Way，1KB-4Way，1KB-2Way，2KB-4Way，2KB-2Way 構成の命令キャッシュに提案手法を適用した場合の命令キャッシュミス率変化をそれぞれ図 3 から 7 に示す．

まず，特筆すべき点として，今回の評価において，キャッシュミス率が提案手法適用により低下することがなかった事があげられる．提案手法は，キャッシュに “入れない” 制御をおこなうことから，キャッシュに入れないと指定されたブロックが，スラッシングの原因とならずに，時間的局所性を持つならば，参照の度に本来発生しないミスを起こす．このことにより，キャッシュミス率が通常実行より増加する可能性が存在する．結果的に，提案手法はこのケースがプロファイルアルゴリズムにより避けられており，安全にバッファ挿入ブロックを選択できたと言える．

図 3 の 512B-2Way 構成では平均で約 27.45%，最大で susan の 53.04%，図 4 の 1KB-4Way 構成では平均で約 25.16%，最大で rijndael の 65.07%，図 5 の 1KB-2Way 構成では平均で約 21.77%，最大で rijn-

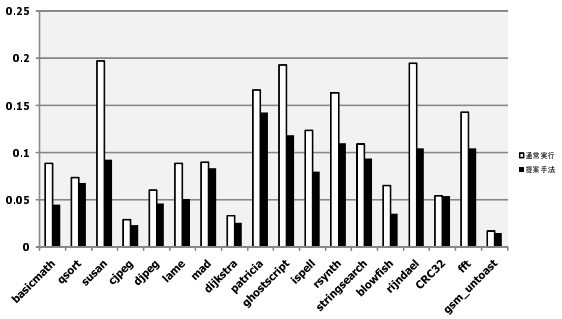

図 3 512B-2Way 構成時の提案手法の効果.

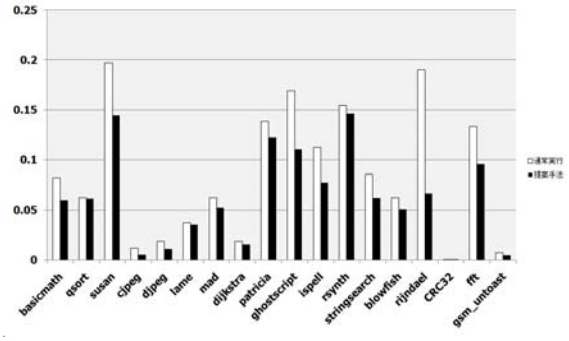

図 4 1KB-4Way 構成時の提案手法の効果.

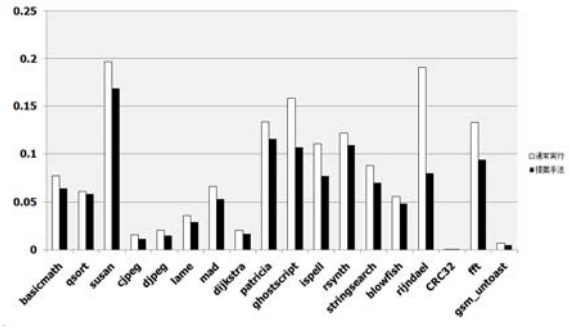

図 5 1KB-2Way 構成時の提案手法の効果.

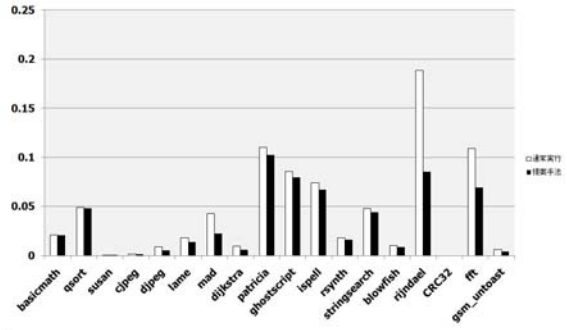

図 6 2KB-4Way 構成時の提案手法の効果.

dael の 57.96%，図 6 の 2KB-4Way 構成では平均で約 18.65%，最大で rijndael の 54.69%，図 7 の 2KB-2Way 構成では平均で約 16.38%，最大で rijndael の 44.98% のキャッシュミス率削減効果となった．全実行の平均は 21.88%のキャッシュミス率削減効果となった．容量と構成が変わると提案手法の効果が低くなるのは，そのキャッシュでスラッシングが発生するセットの数が減るからである．個々のアプリケーションによって傾向は様々であるが，今回計測に使用したデータ上では同一容量であれば，概ねセット数が多い方がスラッシングを避ける傾向にあることがわかった．

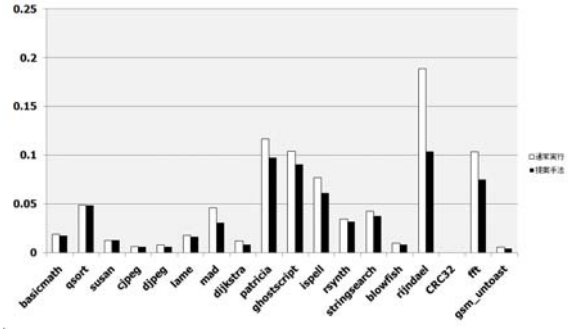

図 7 2KB-2Way 構成時の提案手法の効果.

他の傾向としてはキャッシュ容量が小さいほど，提案手法の効果が大きく見えていることがわかる．今回の評価のプロファイリングアルゴリズムはスラッシング関係にあるブロックを検出するものである．キャッシュ容量が小さいほどスラッシング関係にあるブロックが多くなる可能性が高いことは自明であり，アルゴリズムは各構成に応じて適切に，スラッシング関係にあるブロックを検出できていると言える．

susan や rijindael 等，比較的効果の大きいアプリケーションに対して，構成によって効果が変わる理由としてアルゴリズムのパラメータであるセット数と Way 数の関係が挙げられる．例えば，スラッシング関係のあるブロックが 5 つ有るとして，2Way のキャッシュと 4Way のキャッシュを想定する場合，アルゴリズムによって救えないブロック数が 3 つと 1 つになり差が出る．これがアルゴリズム適用の効果の違いになる．

一方でセットを増やした場合，競合するセットの位置が分散されることから，スラッシング関係が変わり，アルゴリズムで示した最大値の 1/2 の閾値では逆にスラッシング検出できないブロックが増えるケースが存在することがわかった.（512B-2Way と 1KB-2Way の susan のケース）この点に関して，アルゴリズムの洗練が必要とされる．

大きく見た場合，提案手法は全体で 20%以上のキャッシュミス削減効果があることから，現実性を持った手法で実現した場合でも，小容量キャッシュで大きなキャッシュ性能向上を狙える着眼点を持っていると期待できる．

4.3.2 提案手法による命令キャッシュ縮小化の議論

提案手法無し 512B キャッシュのミス率を 512BaseMissRate，提案手法有り 512B キャッシュのミス率を 512PropMissRate，比較対象キャッシュ構成のミス率を TargetMissRate とした場合，他構成キャッシュとの性能差を 512B の提案手法が埋めた比率 R を以下

の式で求めた．

$$R = \frac{512BaseMissRate - 512PropMissRate}{512BaseMissRate - TargetMissRate} \quad (1)$$

それぞれのキャッシュ構成とアプリケーションに対し，TargetMissRate を当てはめ，R を計算した結果を図 8 に示す．

1 に近づく程，基準となる 512B キャッシュと他構成キャッシュの間の性能差が提案手法によって埋まっており，1 を超える場合は他構成キャッシュに対して性能が勝っていることとなる．1 を超える値は性能の逆転を示すのみで，値自体に他の意味を持たないことから，図の縦のスケールは 1 以下をクローズアップしている．2 倍容量のキャッシュに対して，18 アプリケーション中 8 つのアプリケーションで性能が上回っている．4 倍容量のキャッシュ構成に対して，1 を下回るものの平均が，2KB-4Way では約 0.39，2KB-2Way では 0.44 であった．このことから，全体を通して，4 倍容量のキャッシュに対しても，提案手法は性能差の約 4 割を埋めていることが分かる．

5. おわりに

本稿では，組込みプロセッサを対象とし，小規模命令キャッシュのヒット率を向上させる方式を提案した．通常の命令キャッシュと同時に参照される小規模（1 キャッシュブロック相当）のバッファを用意し，当該バッファを有効利用して，命令キャッシュヒット率を向上させるものである．バッファには命令キャッシュミス時に “キャッシュに入れない” データを格納する．また，提案手法はキャッシュに入れるかどうかを選択するハードウェア（FTS）を持ち，FTS に指示を与える方法を示した．

4 節以降の評価では，プロファイルベース・ブロック単位粒度の FTS 制御を理想状態でシミュレーションし，小規模キャッシュにおける提案手法の潜在的性能向上能力を示した．評価の結果，提案手法は平均で約 20%以上のキャッシュミス率削減能力を有することが示された．このことは，今後の組込みシステム分野における小規模命令キャッシュの性能改善の研究に対する明確な動機が示される結果となっている．

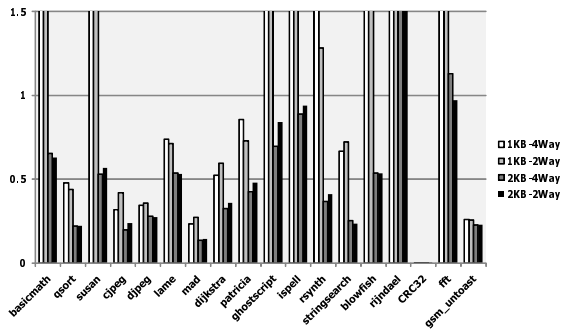

図 8 提案手法の他の構成のキャッシュに対する性能接近率．

今後の課題として，具体的なハードウェア／ソフトウェアの議論が必要であり，その精細な評価を行う予定である．

謝辞 本研究の一部は科学研究費補助金若手研究（B）（20700045）「低消費電力高機能リコンフィギュラブルメモリシステムの研究」の一環として行なわれた．

参 考 文 献

1) ルネサスエレクトロニクス，“SH-2A，SH2A-FPU ソフトウェアマニュアル Rev.3.00”，2005.
2) D. Seal, ARM Limited. “ARM Architecture Reference Manual, 2nd ed.”, Addison-Wesley, 2000.
3) 笹山高志，田中清史，“タスクの優先度を考慮したバイナリ最適化”，組込みシステムシンポジウム 2009 論文集（ESS 2009），pp.127–132，2009.
4) A.Wolfe, A.Chanin, “Executing Compressed Programs on an Embedded RISC Architecture”, Proc. of Intl. Symp. on Microarchitecture, pp.81–91, 1992.
5) C.Lefurgy, P.Bird, I-C.Chen, T.Mudge, “Improving Code Density Using Compression Techniques”, Proc. of Intl. Symp. on Microarchitecture, pp.194–203, 1997.
6) D.Pham, et al., “The Design and Implementation of a First-Generation CELL Processor”, IEEE Intl. Solid-State Circuits Conference, pp.184–185, 2005.
7) J.Montanaro, et al., “A 160-MHz, 32-b, 0.5-W, CMOS RISC Microprocessor”,
8) Intel Corporation, “Intel XScale Microarchitecture, Technical Summary”, 2000.
9) A.Veidenbaum, D.Nicolaescu, “Low Energy, Highly-Associative Cache Design for Embedded Processors”, Proc. of Intl. Conf. on Computer Design (ICCD), pp.332–335, 2004.
10) http://www.simplescalar.com/v4test.html
11) M.Guthaus, J.Ringenberg, D.Ernst, T.Austin, T.Mudge, R.Brown, “MiBench: A free, commercially representative embedded benchmark suite”, Proc. of IEEE Intl. Workshop on Workload Characterization, 2001 (WWC-4).
12) http://www.eecs.umich.edu/mibench/