Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

フィルタ・キャッシュにおける低電力化置換ポリシー

彭 南翔[†1,a)]　安藤 秀樹[†1]

概要：キャッシュの消費エネルギーはプロセッサ全体の消費エネルギーの中で大きな割合を占めている．L1 データ・キャッシュの上に非常に小さなフィルタ・キャッシュを配置することは，キャッシュ全体の消費エネルギーを効果的に削減できる一つの解決策である．
本論文では，フィルタ・キャッシュの特性を考慮して，デッド・ブロック（将来アクセスされることがないブロック）予測とアクセス・ディスタンスを考慮したアンフレンドリ・ブロック（1 度もアクセスされない）予測を組み合わせた置換ポリシーを提案する．SPECCPU2017 を用いて提案手法を評価した．その結果，フィルタ・キャッシュのミス率を LRU に比べて平均 15.1% 削減し，フィルタ・キャッシュ，L1 データ・キャッシュ及び追加したハードウェアの合計の消費電力を平均 5.1% 削減できることを確認した．

1. はじめに

キャッシュの消費エネルギーは大きく，プロセッサが消費する全エネルギーに対して大きな割合を占めている．そのため，キャッシュのエネルギー削減は重要な課題である．現在の多くのマイクロプロセッサは，性能を向上させるために 3 つのレベルのオンチップ・キャッシュを採用しているが，その中でも，L1 データ・キャッシュは頻繁にアクセスされるため消費エネルギーが大きく，これを削減することは重要である．

L1 データ・キャッシュのエネルギーを削減する単純で有効な手法の一つとして，フィルタ・キャッシュがある [1]．フィルタ・キャッシュは，L1 データ・キャッシュの上位に置く非常に小さなキャッシュであり，L0 キャッシュとも呼ばれる．これにヒットした場合，L1 データ・キャッシュにアクセスが行かないので消費エネルギーを削減できる．

しかし，フィルタ・キャッシュは非常に小さいため容量ミスを頻発する．そこで，本研究は，LRU に代わる新しい置換方式を提案し，ミスを減少させることを目的とする．LRU に代わる置換方式として，最終レベル・キャッシュについて多くの研究がなされている [2] [3] [4]．しかし，フィルタ・キャッシュに対する置換方式の研究はなく，フィルタ・キャッシュの特性を考慮した置換方式が必要である．

置換方式を考えるうえで，次の 2 つの性質のブロックについて考慮した．1) デッド・ブロック（キャッシュから追い出される前に，アクセスされることはもはやないブロック），2) リアクセスされない（キャッシュに存在する間，1 度もアクセスされない）ブロック（以下，キャッシュ・アンフレンドリなブロックと呼ぶ）．フィルタ・キャッシュは非常に容量が小さいため，これらのブロックが他のレベルのキャッシュに比べて多く，これらを十分に考慮した方式が有効となる．

1) に対しては，デッド・ブロック予測を導入する．この方式として，カウンター・ベース予測器 [2] を応用する．カウンター・ベース予測器は，キャッシュ・ブロックの過去の参照回

†1 現在，名古屋大学大学院工学研究科
Presently with Graduate School of Engineering, Nagoya Uniersity
a) peng@ando.nuee.nagoya-u.ac.jp

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

数を記録し，それを将来の参照回数と予測するものである．ブロックの参照回数が予測した参照回数に達したら，そのブロックはもはや参照されない（つまり，デッド）と予測する．ミス時の置き換えの際には，LRU ブロックに優先してデッドなブロックを置き換える．

2) については，前述したカウンター・ベース予測を利用したもので，予測器が 1 回しかアクセスしないと予測したブロックをアンフレンドリ・ブロックと予測するものである．ミス時の置き換えの際，このようなアンフレンドリ・ブロックはキャッシュに格納しない，つまりバイパスの候補とする．

本研究は，アクセス・ディスタンス予測と呼ぶ予測を導入し，ブロックが次にアクセスされるまでのディスタンスを予測する．置き換えようとするブロックの次までのアクセス・ディスタンスが，置き換えるセット内のブロックの次までのアクセス・ディスタンスより長いときのみバイパスする．

提案手法を，1KB のフィルタ・キャッシュに対して適用し，SPECCPU2017 ベンチマークを用いて評価した．その結果，全ベンチマークの平均で LRU に対し提案手法は，0.71KB のオーバーヘッドで，最大 54.3%，平均で 15.1% のミス率改善を達成した．また，フィルタ・キャッシュ，L1 キャッシュ及び追加したハードウェアの消費エネルギーは，平均で 5.1%削減できることを確認した．

本論文の以降の構成は以下の通りである．2 章で関連研究について説明し，3 章で本研究で応用するカウンター・ベースのデッド・ブロック予測手法について説明する．4 章で提案手法について説明する．5 章で提案手法について評価を行い，6 章でまとめる．

2. 関連研究

近年の研究では，LRU と理論的に最適な置換アルゴリズムとの間の性能差が大きいことが示されており [2] [5] [6]，キャッシュの性能を向上させるための代替置換アルゴリズムが研究されている．

LRU 置換アルゴリズムでは，ブロックが最後に使用された後 (デッドと定義)，LRU ブロックになるまで長い間キャッシュに留まる．このようなデッド・ブロックは，他のブロックがキャッシュを使用することを不必要に妨げる．

デッド・ブロック予測は，L1 キャッシュと L2 キャッシュのブロックの生死を様々な方法で予測する．この方法は，プリフェッチ，キャッシュ・ブロックの置換，バイパス，消費エネルギーの削減などの方式で活用されている．しかし，フィルタ・キャッシュは非常に小さく，キャッシュ・ブロックがすぐに追い出されてしまうため，予測することが困難である．

3. カウンター・ベース方式

本研究で提案する置換ポリシーは，カウンター・ベースのキャッシュ置換アルゴリズム [2] を基本としている．そこで，本章では，この動作を詳細に説明する．

3.1 構成

カウンター・ベースのキャッシュ置換アルゴリズムの構成について説明する．具体的には，デッド・ブロックの予測とアンフレンドリ・ブロックの予測に，カウンター・ベース方式を利用している．そこで，本章では，このカウンター・ベース方式について詳細に説明する．

図 1 に構成を示す．予測テーブルと呼ぶ構造を用意する．予測テーブルは，タグのない表であり，ハッシュされた PC でインデクスされ，各エントリは，対応するブロックの過去のアクセス数（C_Stored）と信頼ビット（Conf_Stored）を保持する．予測テーブルは，フィルタ・キャッシュ内のブロックがアクセスされる最大回数を予測する．ブロックのアクセス回数が予測アクセス回数に達したとき，そのブロックはデッドになったと予測する．キャッシュ・ミスが発生したら，デッド・ブロックを LRU ブロックに優先して置き換える．

一方，フィルタ・キャッシュには，各ブロックについて以下のフィールドを追加する．

(1) アクセス・カウンター（C_Now）: ブロックのアクセス回数を記録する．アクセスされたら，1 インクリメントする．
(2) アクセス回数の予測値（C_Past）: 予測テーブルにより予測されたアクセス回数を保持する．
(3) 信頼ビット（Conf）: C_Past の予測信頼性を示す．

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

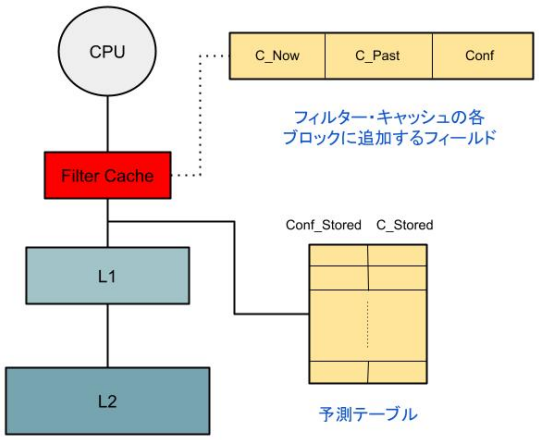


図 1: カウンター・ベース方式の構成

3.2 動作

(1) ヒットの場合：

a) ヒットしたブロックの C_Now をインクリメントする．

(2) ミスの場合：

a) アクセスしたセットにあるすべてのブロックについて，C_Now が C_Past 以上が，また Conf がセットされているかをチェックする．両方成立するブロックがあれば，そのブロックをデット・ブロックと予測する．

b) デット・ブロックがセット内に存在するなら，それを置き換える．デッド・ブロックが複数ある場合，その中からランダムに選択する．デッド・ブロックが存在しなければ，LRU ブロックを置き換える．

c) 置き換えられたブロックの情報を使用して予測テーブルを更新する．つまり， PC をハッシュして予測テーブルをアクセスし，置き換えブロックの C_Now が，アクセスした予測テーブルのエントリの C_Past と等しいなら，当該エントリの Conf_Stored をセットする．そうでなければ，リセットし，C_Now を C_Stored に保存する．

d) 置き換えるブロックをキャッシュに挿入し，そのブロックの予測アクセス回数に関する情報を以下のように初期化する:

C_Now = 0，C_Past = C_Stored，
Conf = Conf_Stored

カウンター・ベースの置換方式は，5 章で評価する．

4. 提案手法

本章では提案手法について詳しく説明する．

4.1 キャッシュ・バイパス手法の提案

カウンタ・ベース方式の予測テーブルで予測されたアクセス回数が 1 のブロックをアンフレンドリと予測し，このようなブロックをキャッシュ・バイパスすることが有効と考えられる．しかし，一方，カウンター・ベース方式によりデッドと予測されたブロックもアンフレンドリ・ブロックと同様リアクセスされないと予測されるので，デッド・ブロックをキャッシュの残すかアンフレンドリ・ブロックをキャッシュに書き込むかを選択する必要がある．

上記の課題を解決するために，Belady の OPT アルゴリズム [7] を考慮した．具体的には，以下のように修正して，バイパスにおいて利用する．

> 最適な置き換えブロックは，セット内の各ブロックについて，次のアクセスまでディスタンスが最大の値を持つブロックである．

この修正は，次のアクセスまでディスタンスが最も小さいブロックは，将来も再現されるという仮定に基づいている．

これを考慮し，前述した課題について，以下のように対処する．ビクティム・ブロック（置き換えようとするブロック）とアンフレンドリ・ブロックの次のアクセスまでの予測されるディスタンスを比較し，後者のディスタンスが前者より大きければ，バイパスを行う．そうではなければ，置換を行う．

ここで，ディスタンスとは，フィルタ・キャッシュでのアクセス・バースト数と定義する．バーストは，ブロックが LRU スタックの MRU 位置に移動したときに始まり，他のブロックが MRU 位置に入り，元のブロックが MRU 位置から下がったときに終わる．アクセス・ディスタンスの予測は，フィルタ・キャッシュでのアクセス・バースト数を数えることにより行う．

4.2 構成

提案手法の全体構成を図 **2** に示す．カウンタ・

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

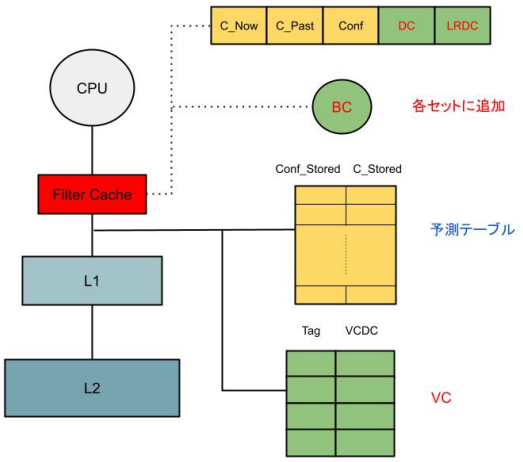


図 2: 提案手法の実装

ベース方式の構成（図 1）をベースに，バイパスのためのハードウェアを追加している．

アクセス・ディスタンスは，フィルタ・キャッシュにおいて学習するほか，フィルタ・キャッシュの下にビクティム・キャッシュ (VC: victim cache) [8] と呼ぶ追加の小さなキャッシュを追加し，ビクティム・ブロックおよびバイパスしたブロックのアクセス・ディスタンスを学習する．VC には，ビクティム・ブロックあるいはバイパスされたブロックのタグとディスタンスの値を挿入する．VC を加えたのは，バイパス・ブロックはフィルタ・キャッシュに格納されないので，アクセス・ディスタンスを学習できないからである．

アクセス・ディスタンスを学習するため，フィルタ・キャッシュの各ブロックに 2 つの飽和型のカウンター（distance counter(DC) と last reaccess distance counter(LRDC)）を追加する．

VC へのアクセスを削減するため，各セットにバースト・カウンター（BC）と呼ぶカウンタを設置している．

以下，それぞれについて説明する．

4.2.1 フィルタ・キャッシュの追加フィールド

フィルタ・キャッシュの各ブロックに以下のフィールドを追加する．

(1) Distance Counter (DC: 4 ビット)

(2) Last Reaccess Distance Counter (LRDC: 4 ビット)

DC は飽和型のカウンターで，ブロックの前回アクセスから経過したディスタンスを測定する．LRDC も飽和型のカウンターで，DC によりリアクセス・ディスタンスが得られたら（すなわち，対応するブロックへの再アクセスがあったなら），最も最近のリアクセス・ディスタンスとして記憶する．

まだ，フィルタ・キャッシュの各セットに以下のフィールドを追加する．

Burst Counter(BC: 4 ビット)

BC は飽和型のカウンターで，セットにおける最近のミス発生時点から，現在までのディスタンス (バースト数) を測定する．

BC は電力削減ための重要なハードウェアである．BC がなければ，ディスタンスが増加する度に，VC をアクセスして VCDC を更新する必要がある．BC を使用して，ヒット時に更新されるディスタンスを記録し，ミス時だけその値を VC に転送する．これにより，容量が大きい VC へのヒット時のアクセスを回避し，消費電力の削減を実現している．

4.2.2 VC によるアクセス・ディスタンスの学習

バイパス・ブロックおよびフィルタ・キャッシュから追い出されたブロックについてリアクセス・ディスタンスを学習する．以下のフィールドを持つ．

(1) Tag (8 ビット)

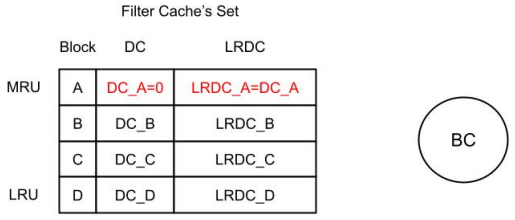
(2) Victim Cache's Distance Counter (VCDC:4 ビット)

Tag はキャッシュの通常のタグと同じであるが，ハッシュすることにより 8 ビットに圧縮する．VCDC はフィルタ・キャッシュの DC と同じように，最も最近のアクセスから現在のアクセスまでのディスタンスを保持する．VC のセット数はキャッシュのセット数と同じであり，各セットは，フィルタ・キャッシュのセットに対応している．置換方式は LRU である．

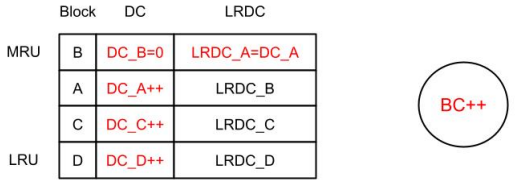
4.3 動作

提案手法の動作は，フィルタ・キャッシュの情報の更新，VC の更新，置換とバイパスの 3 つの部分から構成される．デッド・ブロックの予測に関する動作については， 3.2 節で説明している．ここでは省略する．

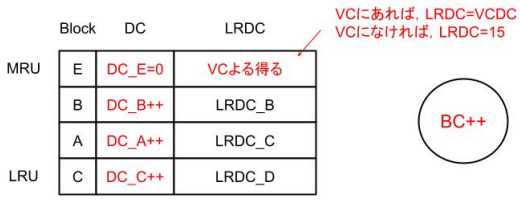
Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10



(a) ヒット，A をアクセス

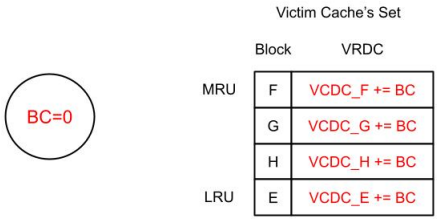


(b) ヒット，B をアクセス

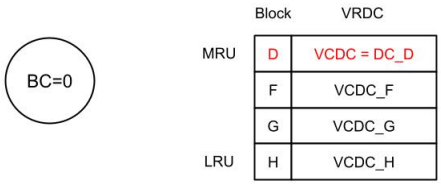


(c) ミス，E をアクセス, D を追い出す

図 3: フィルタ・キャッシュの情報の更新



(a) BC と VCDC の更新



(b) ミスしたブロックが E，追い出されたブロックが D の場合

図 4: VC の動作

4.3.1 フィルタ・キャッシュの情報の更新

フィルタ・キャッシュの情報の更新について，**図 3** を用いて説明する．

1) ヒットの場合

- 図 3 の (a) : ヒットしたブロック A について，DC によりリアクセス・ディスタンスが得られる．これをリアクセス・ディスタンスの予測値とし，LRDC にコピーする．そして，次のディスタンスを数えるため，DC を 0 にクリアする．MRU にあるブロックにヒットしたので，バーストは終了しない．このため，ディスタンスは変わらず, 各ブロックの DC の値はそのままである．
- 図 3 の (b) : ヒットしたブロック B が MRU にないので，現在のバーストが終わる．このため，ブロック B を除くセット内の各ブロックの DC を 1 増加する．また，BC もインクリメントする．

2) ミスの場合

- 図 3 の (c) : ミスしたセットにあるすべてのブロックについて，DC を 1 増加する．BC も増加する．置換を行った後，挿入したブロック E の DC をリセットする． VC をアクセスし，ブロック E を保持しているなら，VC の VCDC をフィルタ・キャッシュの LRDC にコピーする．保持していなければ，LRDC を最大値 (4 ビットなので 15) にセットする．さらに，VC がブロック E を保持している場合，VC のブロック E を削除する．

4.3.2 VC の動作

VC の動作を**図 4** を使って説明する．VC は，フィルタ・キャッシュへのアクセスがミスしたときだけ動作する．

1) BC と VCDC の更新 (図 4 の (a)) : バーストの切れ目が現れたら，BC をインクリメントすることによりディスタンスを更新する． フィルタ・キャッシュにミスしたブロックに対応するセットにあるすべてのブロックの VCDC に BC を加算する．その後，次のディスタン

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

スを数えるため BC を 0 にクリアする．

2) 後述したようにバイパスすると判断した場合，そのブロックを VC に入れて，VCDC を 0 にクリアする．

3) フィルタ・キャッシュの置換時 (図 4 の (b)): フィルタ・キャッシュのビクティム・ブロックを VC に入れて，その上で，当該ブロックの前回アクセスから経過したディスタンスを継続的に学習するために，DC の値を VCDC にコピーする．

4) バイパス時：バイパスするブロックを VC に入れて，VCDC を 0 にクリアする．

4.3.3 置換とバイパス

フィルタ・キャッシュ・ミスが生じた場合の，置換するかあるいはバイパスするかの判断の流れを，**図 5** に示す．以下，詳細な説明を行う．

1) ビクティム・ブロックの選択：カウンター・ベース方式の予測により，ビクティム・ブロックを選択する．つまり，関連するセットにデッド・ブロックがある場合，デッド・ブロックを選択する．そうでない場合，LRU ブロックを選択する．

2) ミスしたブロックがアンフレンドリかどうかの判断 (図 5 の 1)): カウンター・ベース方式の予測テーブルにより，ミスしたブロックがアンフレンドリと予測されたら，次に進む．そうでなければ，リアクセスされると予測してミスしたブロックに置換する．

3) ビクティム・ブロックの次のアクセスまでのディスタンス予測 (図 5 の 2)): ビクティム・ブロックについて，DC>LRDC の場合，現在のディスタンスは前回のそれより大きいため予測は無効とし，置換する．そうでなければ，次のアクセスまでのディスタンスを，LRDC−DC により計算する．

4) アンフレンドリなブロックの次のアクセスまでのディスタンス予測: VC にアンフレンドリ・ブロックを探し，VC に存在したら，そのフィールドにある VCDC をリアクセス・ディスタンスの予測値とする．アンフレンドリ・ブロックが VC に存在しなければ，リアクセス・ディスタンスは非常に長いことを示す．この場合，最大値 15 を予測値とする．

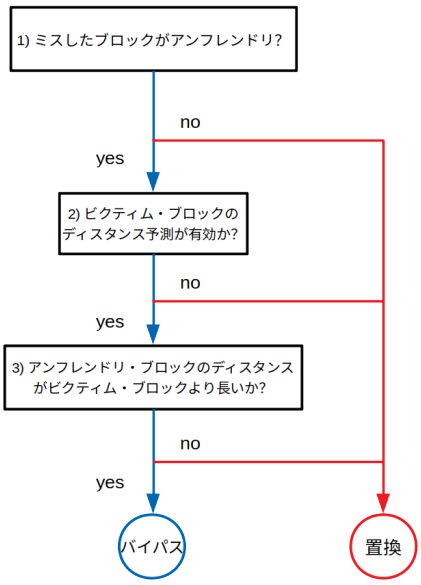


図 5: 置換とバイパスの流れ

5) 次のアクセスまでのディスタンスの比較 (図 5 の 3)) : ビクティム・ブロックの次のアクセスまでのディスタンス (step(3) による予測) がアンフレンドリなブロックの次のアクセスまでのディスタンス (step(4) による予測) より大きいなら，置換を行う．そうではなければ，バイパスする．

5. 評価

5.1 評価環境

評価環境について説明する．シミュレータには SimpleScalar [9] をベースに修正を加えたものを使用した．評価には，SPECCPU2017 ベンチマークを使用した．ベンチマークの測定区間は，プログラムの先頭から 10B 命令をスキップした後の 100M 命令とした．評価のベースは LRU を用いる 1KB のフィルタ・キャッシュを実装したモデルである．

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

5.2 ハードウェア・コスト

(1) 予測テーブル:予測テーブルは，1024 エントリ，各エントリは 5 ビット（有効ビット 1 ビット，記録するアクセス回数 4 ビット）とした．したがって，予測テーブルの容量は 1024 × 5=5120 ビット．

(2) VC:VC は 4 セット，各セットが 8 エントリ，各エントリが 12 ビット（タグ 8 ビット，VCDC4 ビット）とした．したがって，VC の容量は 4 × 8 × 12=384 ビット．

(3) フィルタ・キャッシュの追加容量：フィルタ・キャッシュは 16 ブロックがあり，各ブロックに追加するビットが 17 ビットである．この 17 ビットの内訳は，C_Now(4 ビット)，C_Past(4 ビット)，Conf(1 ビット)，LRDC(4 ビット)，DC(4 ビット) である．さらに，フィルタ・キャッシュは 4 セットがあり，各セットに 4 ビットの BC を追加する．したがって，フィルタ・キャッシュの追加容量は 16 × 17+4 × 4=288 ビット．

以上より，提案手法はおよそ 0.71KB（5792 ビット）のオーバーヘッドで構成することができる．

5.3 フィルタ・キャッシュのミス率

カウンター・ベースのモデル及び提案手法のフィルタ・キャッシュ・ミス率を**図 6** に示す．縦軸は，ベースで規格化したミス率を示す．

カウンター・ベースモデルのベースで規格化したミス率は平均で 88.0%であった．提案手法の場合，ベースで規格化したミス率は平均で 84.9%であった．つまり，提案手法によりミス率を LRU から 15.1%削減できた．その中，roms のミス削減率は最も高く，53.4%である．これで，フィルタ・キャッシュのミスの削減することに対して，提案手法はカウンター・ベースの方式より有効と言える．

5.4 消費エネルギー

カウンター・ベースのモデル及び提案手法のフィルタ・キャッシュと L1 キャッシュの合計の消費エネルギーを評価する，提案手法の消費エネルギーには，追加ハードウェアの消費エネルギーも含まれている．評価には，McPAT [10] を用いた．**図 7** に評価結果を示す．縦軸は，ベースで規格化した消費エネルギーである．

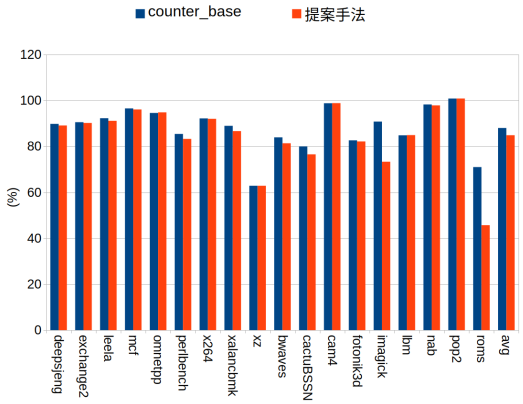


図 6: フィルタ・キャッシュのミス率

カウンター・ベースの方式は平均で 2.8%の消費エネルギー削減する．これに対して，提案手法では，ベースに対する消費エネルギーは平均で 94.9%となり．つまり，5.1%の消費エネルギー削減を達成している．

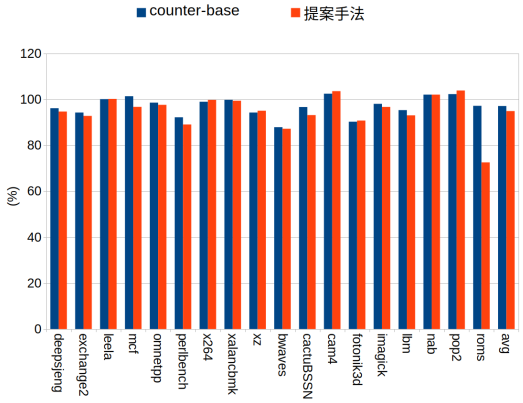


図 7: ベースで規格化したフィルタ・キャッシュ，L1 キャッシュ及び予測機構の合計消費エネルギー

図 8 に，提案手法の各コンポーネントの消費エネルギーの内訳を示す．縦軸は，ベースのフィルタ・キャッシュおよび L1 キャッシュの消費エネルギーの合計で規格化している．VC，pred_table は，それぞれ，VC，予測テーブルの消費エネルギーを示す．平均では，VC と予測テーブルの合計消費エネルギーは全体の消費エネルギーの 4.2%しかなく，手法のオーバヘッドは小さいといえる．

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

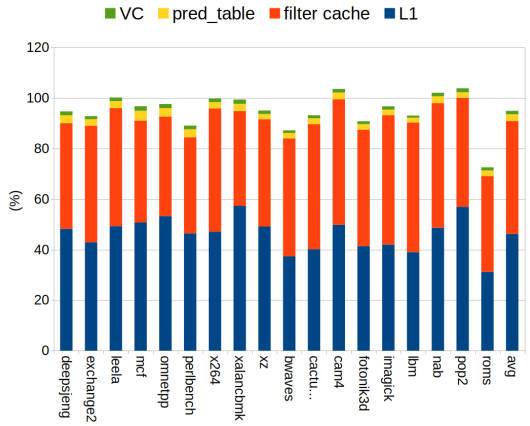


図 8: ベースで規格化した提案手法の消費エネルギーの内訳

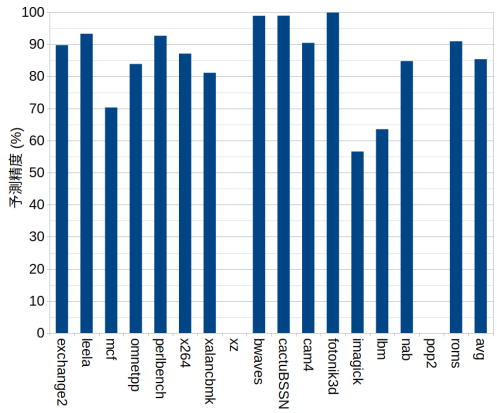


図 9: アクセス・ディスタンスの予測精度

5.5 アクセス・ディスタンスの予測精度

ここでいう予測精度とは，4.3.3 節の 5) に記述する次のアクセスまでのディスタンスの比較が正しいかどうかということである．

シミュレータにコードを追加することで，リアクセス・ディスタンスの比較を行ったブロックを追跡して，その予測精度を評価した．**図 9** に測定結果を示す．同図からわかるように，平均な予測精度は 85.3%で (xz と pop2 は予測することがないために含めない)，ミス率の削減に貢献する十分な精度を持つと考える．fotonik3d と roms の精度は 99.9%と最も高い．

6. まとめ

本研究は，キャッシュのミス率と消費エネルギーを削減するため，小さなフィルタ・キャッシュに対する置換ポリシーを提案した．

SPECCPU2017 ベンチマークを用いて評価を行った結果，提案手法を導入することにより，LRU と比較して 15.1%削減できることを確認した．これにより，LRU と比較して，フィルタ・キャッシュ，L1 キャッシュと追加したハードウェアの合計の消費エネルギーを 5.1%削減できた．

参考文献

[1] J. Kin, M. Gupta, and W. Mangione-Smith, “The filter cache: an energy efficient memory structure,” in *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, December 1997, pp. 184–193.

[2] M. Kharbutli and Y. Solihin, “Counter-based cache replacement and bypassing algorithms,” *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 433–447, February 2008.

[3] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, “The evicted-address filter: A unified mechanism to address both cache pollution and thrashing,” in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2012, pp. 355–366.

[4] R. Karedla, J. S. Love, and B. G. Wherry, “Caching strategies to improve disk system performance,” *Computer*, vol. 27, no. 3, pp. 38–46, March 1994.

[5] J. H. H. Liu, M. Ferdman and D. C. Burger, “Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency,” in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, November 2008, pp. 222–333.

[6] A. Jain and C. Lin, “Back to the future: Leveraging Belady’s algorithm for improved cache replacement,” in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, August 2016, pp. 78–89.

[7] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[8] N. P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers,” in *Proceedings of the 17th Annual International Symposia on Computer Architecture (ISCA)*, May 1990, pp. 364–373.

[9] D. Burger and T. M. Austin, “The simplescalar tool set, version 2.0,” *ACM SIGARCH Computer Architecture News*, vol. 25, no. 3, pp. 13–25, June 1997.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT:

Vol.2022-ARC-248 No.18
Vol.2022-SLDM-198 No.18
Vol.2022-EMB-59 No.18
2022/3/10

An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, December 2009, pp. 469–480.