Vol.2016-ARC-219 No.13
Vol.2016-SLDM-175 No.13
Vol.2016-EMB-40 No.13
2016/3/24

最適な置き換え予測に基づくフィルタ・キャッシュのミス率削減

古橋 陽平[†1,a)]　塩谷 亮太[†1]　安藤 秀樹[†1]

概要：イン・オーダのプロセッサでは，キャッシュ・レイテンシが性能に大きな影響を与える．キャッシュ・レイテンシを短縮する方法として，フィルタ・キャッシュがある．しかし，フィルタ・キャッシュは非常に小さいため，従来の LRU 置換アルゴリズムでは頻繁に容量ミスを起こし，高いヒット率が得られない．本論文では，フィルタ・キャッシュのミス率を削減する手法として，Optimal 置換アルゴリズムを予測により擬似的に実現する手法を提案する．Optimal 置換アルゴリズムは最適な置き換えを行う理想的なアルゴリズムである．しかし，ラインが将来アクセスされるまでの時間を使用するため実際には実現できない．これに対し，提案手法ではこの将来アクセスされるまでの時間を予測することで，Optimal 置換アルゴリズムと同様の置き換えを行う．SPEC2006 ベンチマークで評価した結果，本アルゴリズムでは LRU 置換アルゴリズムに比べてキャッシュ・ミス率を 17.9%削減することができた．

1. はじめに

イン・オーダ・プロセッサ（InO プロセッサ）は，性能よりも消費電力が重視される場合に主に使用される．たとえば，近年の Internet of Things（IoT）やウェアラブルなどの非常に小型なデバイスでは電力の制約が厳しく，InO プロセッサが使用されることが多い．InO プロセッサが使用される別の例としては，アウト・オブ・オーダ・プロセッサ（OoO プロセッサ）と InO プロセッサを組み合わせた ARM big.LITTLE アーキテクチャがある．big.LITTLE では性能と電力効率の重要性に応じて OoO/InO プロセッサを使い分けることにより，性能と電力効率の両立をはかっている．これらで使用される商用の InO プロセッサとしては，ARM Cortex-A35[1]，Cortex-M7[2]，Cortex-A53[3] などがある．このようなプロセッサでは低消費電力であることに加え，近年ではユーザ要求の高まりから高性能化への要求も高まっている．

このような InO プロセッサの性能には，キャッシュ・レイテンシが大きな影響を与える．一般に OoO プロセッサではキャッシュ・レイテンシは OoO 実行により隠蔽されるため，その影響は小さい．しかし，InO プロセッサではレイテンシを隠蔽することができず，直接的に性能を低下させる．近年のプロセッサではキャッシュ・レイテンシは増加する傾向にあり，L1 データ・キャッシュ（L1 D キャッシュ）のアクセスにも数サイクルを要することから，性能への影響は大きい．たとえば ARM Cortex-A35 や Cortex-A53 では， L1 D キャッシュへのアクセスに 2 サイクル [1], [3]，Cortex-M7 では 3 サイクル [2] を要する．このようなレイテンシの増加は，プロセスの微細化により配線遅延の影響が大きくなっているためである．

L1 D キャッシュとプロセッサとの間に非常に小容量なキャッシュを配置する**フィルタ・キャッシュ (FC)** という手法が提案されている [4]．FC は非常に小さいことから低消費電力で動作でき，電力効率を改善することができる．本論文では，キャッシュのレイテンシの削減を行うために，この FC を使用する．FC は非常に小さいため配線遅延の影響が小さく，低レイテンシで動作できる．メモリ・サブシステムに FC を追加することにより，平均キャッシュ・レイテンシを削減できる．FC の問題点は，非常に小容量であることから Least Recently Used（LRU）に基づく従来の置換がうまく働かないことである．

これに対し本論文では，ごく小容量な FC 向けの，**Pseudo-OPT (POPT) 置換アルゴリズム**を提案する．POPT 置換アルゴリズムは，**Optimal (OPT) 置換アルゴリズム**を予測により擬似的に実現する．OPT 置換アルゴリズムは，最もミスが少なくなる理想的なアルゴリズムである [5], [6]．OPT 置換アルゴリズムでは，キャッシュ・ミスした際に置換対象のキャッシュ・ラインの中で将来のアクセスが最も遅いキャッシュ・ラインを追い出す．将来

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

のアクセス時刻を知る必要があることから，OPT 置換アルゴリズムは実際には実現不可能である．これに対し提案手法では，この次回アクセス時刻を予測し，それに基づき OPT アルゴリズムと同様の置き換えを行う．提案する予測器により次回アクセス時刻を高精度に予測できるため，OPT 置換アルゴリズムに近い動作を行う事が可能であり，キャッシュ・ミス率を削減することができる．提案手法を 256B の FC で評価した結果，全ベンチマークの平均で LRU 置換アルゴリズムに対し 17.9%のミスを削減し，OPT 置換アルゴリズムに対しミス数が 20%差となるところまで迫ることができた．

本論文の以降の構成は以下の通りである．2 章では関連研究について説明し，3 章で提案手法について説明する．4 章で提案手法について評価を行い，5 章でまとめる．

2. 関連研究

LRU 以外のキャッシュ管理アルゴリズムとして，様々な手法が提案されている [7], [8], [9], [10], [11], [12], [13], [14], [15].

これらの中でも，デッド・ブロック予測は，ラインの将来のアクセスを予測しキャッシュ管理をするという点で提案手法と近い．デッド・ブロック予測では，L1 D キャッシュや L2 キャッシュにおいて，様々な方法でキャッシュ内のラインの生死を予測する．長期間アクセスされないと予測したラインを優先的に置換，あるいは今後もアクセスされると予測したラインを保持し続けることで，ミス率の改善や消費電力の削減をする．PC やメモリ・アドレスに基づくシーケンス・ベース [7], [8]，ラインがアクセスされてからの時間に基づく時間ベース [9], [10]，キャッシュやラインへのアクセス回数に基づくカウンタ・ベース [11]，バースト・アクセスの回数に基づくバースト・ベース [12] などのデッド・ブロック予測が存在する．FC は非常に小さく，キャッシュ・ラインがすぐに追い出されてしまうため，FC のミス率削減には使用できない．

3. 提案手法

本論文では，FC のミス率削減のために **Pseudo-OPT (POPT) 置換アルゴリズム**を提案する．提案手法では，OPT 置換アルゴリズムで必要な「各キャッシュ・ラインが次回いつアクセスされるか」を予測することで，置き換えを行う．以下では，提案手法の構成，動作を順に説明する．

3.1 構成

本手法では，以下の構成が必要となる．

(1) FC の各ラインへ追加フィールド: 学習をし予測結果を保持・更新する．
(2) **Victim Buffer (VB)**: FC から追い出されたラインについて学習する．
(3) **Next Access Predicting Table (NAPT)**: 測定したアクセス間隔を保存し，次回のアクセスを予測する．

以下でそれぞれ順に詳しく説明する．

3.1.1 フィルタ・キャッシュの追加フィールド

FC では，同一キャッシュ・ラインへのアクセスの間隔を測定し，予測した次回アクセスまでの間隔を保持・更新する．これを行うために，従来の FC の各キャッシュ・ラインに以下の追加フィールドが必要である．

(1) Delta Access Training Counter (DATC)
(2) NAPT-Index Field (NIF)
(3) Count-Down Timer (CDT)
(4) Prediction Valid Bit (PVB)

DATC は飽和型アップ・カウンタで，前回のアクセスからの間隔を数える．本手法でのアクセスの間隔には，キャッシュ・アクセスの数を使用する．NIF は，NAPT へのインデクスを保持する．測定したアクセス間隔を NAPT へ保存する際に用いる．CDT は飽和型ダウン・カウンタで，次回のアクセスまでの間隔の予測値を保持・更新する．この CDT には，負の値も保持できる飽和型ダウン・カウンタを用いる．PVB は，CDT の値が有効であることを示すビットである．

3.1.2 Victim Buffer (VB)

VB は，上記の FC の追加フィールドのうち DATC と NIF のみを保持し，FC から追い出されたラインについて学習を継続する．この表は CAM で構成し，エントリは LRU で置き換える．タグには，ライン・アドレスを用いる．

3.1.3 Next Access Predicting Table (NAPT)

NAPT では，FC や VB を使って測定したアクセス間隔を保存し，次回アクセスのタイミングを予測する．各エントリは，保存したアクセス間隔と有効ビットからなる．

インデクスには，直前にそのラインへアクセスしたメモリ命令の PC と，アクセスしたライン内オフセットとを XOR したものを用いる．一方で，予測のインデクスにメモリ・アドレスを使用している従来のデッド・ブロック予測では，メモリ・アドレスのうちライン・アドレスを使用している手法がほとんどである [8], [11], [12].

3.2 動作

本手法は，キャッシュ・ラインが次にいつアクセスされるかを予測することにより OPT 置換アルゴリズムを擬似的に実現する．**図 1** に POPT 置換アルゴリズムの動作の様子を示す．まず，DATC により同一キャッシュ・ラインへのアクセスの間隔を学習する (①)．次に，NAPT を用いて次のアクセスがどれだけ遅いかの予測を行い (②)，予測結果はライン内の CDT で保持・更新する．ミスをした際には，この予測結果に基づいて置換を行う (③)．以下では，学習，予測，置換のそれぞれについて動作を説明する．

Vol.2016-ARC-219 No.13
Vol.2016-SLDM-175 No.13
Vol.2016-EMB-40 No.13
2016/3/24

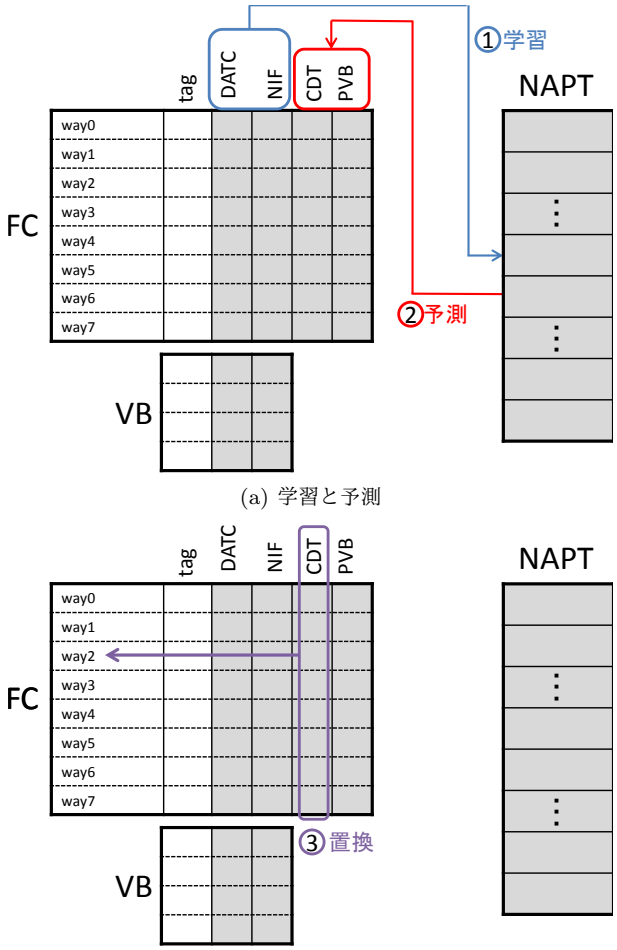


図 1　POPT 置換アルゴリズムの動作

3.2.1　次回アクセスまでの間隔の学習

アクセス間隔の学習 (図 1 内①) は，あるラインへアクセスがあった場合に，同一ラインへの次のアクセスまでのキャッシュ・アクセスの数を数え，記録する．アクセス間隔の測定は FC と VB の DATC で行う．アクセスされたラインが FC 内に存在すれば FC で測定し，そうでなければ VB により測定する．測定結果は，NAPT へ記録する．

アクセス間隔の学習は，具体的には DATC と NAPT を用いて以下のように行う．

(1) あるラインがアクセスされたら，対応する DATC を 0 にリセットする．同時に，PC とライン内オフセットから NAPT のインデクスを生成し，ライン内の NIF へ保存する．後に (3) で，この時保存した NAPT のインデクスを用いて，測定結果を保存する．

(2) 自身以外のラインがアクセスされるたびに，DATC をインクリメントする．

(3) 次にそのラインがアクセスされたら，(1) で保存した NIF の値をインデクスとして NAPT の対応するエントリへ DATC の値を記録する．これと同時に (1) へ戻り繰り返す．

この動作の例を図 **2** を用いて説明する．図左側の上と下

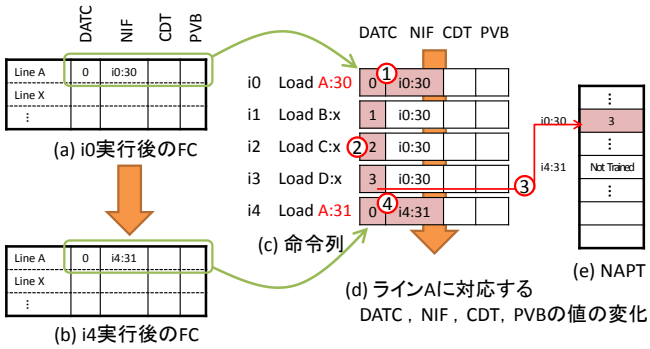


図 2　学習の動作例

にある (a), (b) は，ある時刻での FC 内のライン A に対応した DATC, NIF のそれぞれの値を表す．図中央の (c) は動的命令列の例で，各命令は A から D で表されるラインへアクセスするロード命令である．アルファベットの後の数字はライン内オフセットを表す．例えば，A:30 はライン A の 30 バイト目を表す．(c) の右側の (d) は FC 内のライン A に対応する DATC, NIF の値の変化を (c) の命令列と対応させて示した．(d) の最初と最後の状態は，それぞれ (a)(b) に対応する．図右側の (e) は NAPT を表す．図 2 では，以下のようにアクセス間隔を学習する．

① : ライン A へのアクセスで，ライン A に対応する DATC を 0 にリセットする．これと同時に，NAPT へのインデクスを i0 の PC とライン内オフセットを用いて生成し (i0:30)，NIF へ記録する．ここで記録したインデクスは，③で測定結果を NAPT へ記録する際に使用する．

② : A 以外のラインへのアクセス毎に，A に対応する DATC をインクリメントする．

③ : ライン A への再度のアクセスで，①で NIF へ記録しておいた NAPT のインデクスを用いて，NAPT へ DATC の値 (3) を記録する．

④ : DATC の値を NAPT へ記録したら，①と同様に DATC を 0 にリセットし，NAPT のインデクスを生成し NIF へ記録する．この後，ライン A は長期間アクセスされないとすると，i4:31 のインデクスに対応する NAPT エントリは未学習のままとなる．

以上のような動作を繰り返すことで，DATC によりアクセス間隔を測定し，NIF を用いて NAPT へ記録できる．

キャッシュ・ラインが FC から追い出された場合には，引き続き VB でアクセス間隔を測定する．ラインが追い出される時に，VB のエントリを確保する．ライン内の DATC と NIF の値を，VB のエントリへ移動する．VB でも引き続き，他のラインへのアクセスで DATC をインクリメントする．VB でそのラインへのアクセスを検出したら，FC と同様に NIF を用いて測定結果を NAPT へ記録する．

3.2.2　次回のアクセスまでの間隔の予測

ラインへアクセスが生じたら，次回アクセスまでの間隔

Vol.2016-ARC-219 No.13
Vol.2016-SLDM-175 No.13
Vol.2016-EMB-40 No.13
2016/3/24

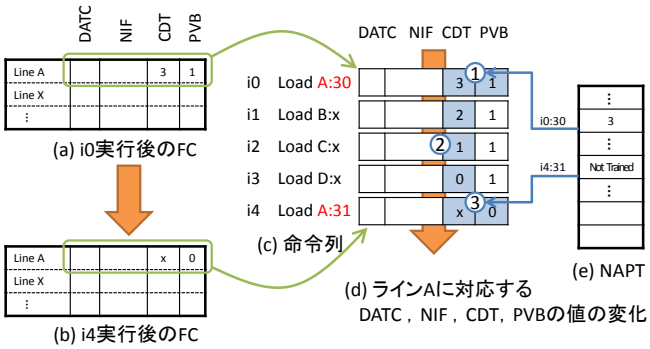


図 3 予測の動作例

を予測する．このために，アクセスしたメモリ命令の PC とライン内オフセットに基づいて NAPT へアクセスし，NAPT の対応するエントリから次回のアクセスまでの間隔の予測値を得る．この予測値をライン内の CDT へ記録する (図 1 内②)．

CDT で保持している次回アクセスまでの間隔は，キャッシュ・アクセス毎に更新する．自分以外のラインへのアクセスで，CDT をデクリメントする．

この動作を**図 3** の例を用いて説明する．図の見方は図 2 と同じである．ただし (a)(b)(d) では，DATC と NIF ではなく，CDT と PVB の値を示した．過去に図 2 の例に示すように学習をしていれば，同じ命令列を実行するときには NAPT は図 3 のように学習がされている．

① : ライン A へのアクセスで，アクセスしたメモリ命令 i0 の PC とライン内オフセット (30) から NAPT インデクスを生成 (i0:30) し，NAPT より次回アクセスまでの間隔を予測する．予測結果である 3 をライン A に対応する CDT へ記録し，PVB を 1 にセットする．

② : A 以外のラインへのアクセス毎に，ライン A に対応する CDT の値をデクリメントする．後述する置換の際には，この CDT の値に基づいて置き換える．

③ : ライン A へのアクセスで，①と同様に，NAPT インデクスを生成 (i0:31) し，NAPT より次回アクセスまでの間隔を予測する．今回のように，NAPT の対応するエントリが未学習である場合には，PVB を 0 にリセットすることで予測されていないことを記録する．

予測のインデクスにライン内オフセットを用いることで，図 3 の i4 のような今後長期間アクセスされないアクセスを識別できる．例のような，ライン内を 1 バイトずつ順にアクセスするメモリ・アクセス・パターンでは，最後の 1 バイト以外での次回アクセスまでの間隔は短いが，最後の 1 バイトの次回アクセスまでの間隔は非常に長い．インデクスにオフセットを含めることで，これらのアクセスを識別することができる．

3.2.3 置換

キャッシュ・ミスが生じたら，前節で述べた予測結果に基づき，どのキャッシュ・ラインを追い出すかを決定する (図 1 内③)．本手法では，基本的には，次回アクセスが最も遅いと予測されるキャッシュ・ラインを追い出す．しかし，予測ができなかった場合など例外があり，具体的には以下の優先順位で置換を行う．

(1) 次回アクセスが予測できないキャッシュ・ライン (図 3(b) の Line A)
次回アクセスまでの予測ができないのは，NAPT に学習結果がなく，今までにアクセス間隔を測定できていない場合である．FC や VB でアクセス間隔の測定ができていないということは，次回アクセスまでの間隔が非常に長いことを意味する．そのため，予測ができないキャッシュ・ラインは，次回アクセスが非常に遅いと予測されたとして扱う．したがって，次回アクセスまでの間隔が予測できなかったキャッシュ・ラインは，最優先で追い出す．

(2) CDT の値が-2 以下のライン
予測した次回アクセスのタイミングを過ぎると，CDT の値は負となる．次回アクセスまでの間隔の予測値が最大であるラインを追い出すため，CDT の値が負のラインは追い出されにくい．予測の誤りによりこのようなラインが多く存在すれば，FC の容量を無駄にしてしまう．FC の容量の無駄を減らすために，このようなラインは優先的に追い出す．ただし，予測の僅かなずれを許容するために 1 回のキャッシュ・アクセスの猶予を設け，CDT の値が -2 以下となったラインは優先的に追い出す．

(3) 置換候補のキャッシュ・ラインのうち，最も次回アクセスが遅いと予測されるキャッシュ・ライン
基本的には，最も次回アクセスが遅いと予測されるキャッシュ・ラインを追い出す．まず，ミスをしたラインの次回のアクセスまでの間隔を，前述したように NAPT を用いて予測する．次に，全ラインの次回アクセスまでの間隔を比較し，次回アクセスが最も遅いキャッシュ・ラインを追い出す．つまり，FC 内の全ラインのそれぞれの CDT の値と，先ほど予測したミスをしたラインの次回アクセスまでの間隔の予測値との中で，最も値が大きいキャッシュ・ラインを追い出す．もしも，ミスをしたラインの次回アクセスを追い出すと選択した場合には，そのラインは FC へ挿入しない．

追い出すキャッシュ・ラインを選択する実際の回路は，トーナメント形式で決定する．**図 4** に示すように，トーナメント形式で最も次回アクセスの遅いラインを選択する．図内の丸印では，図の右側のように次回のアクセスまでの間隔を比較し，値の大きかった側を次段へ送る．

以上のように動作させることで，次回アクセスの予測により OPT 置換アルゴリズムを擬似的に実現できる．OPT 置換アルゴリズムを擬似的に実現することで，キャッシュ・

Vol.2016-ARC-219 No.13
Vol.2016-SLDM-175 No.13
Vol.2016-EMB-40 No.13
2016/3/24

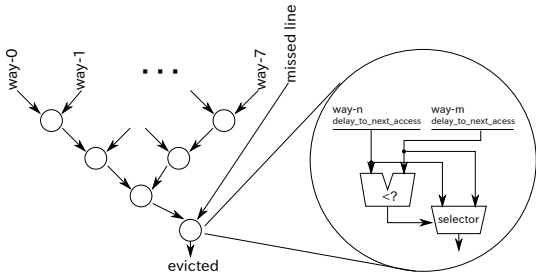


図 4　POPT 置換アルゴリズムで追い出すキャッシュ・ラインを選択する実装

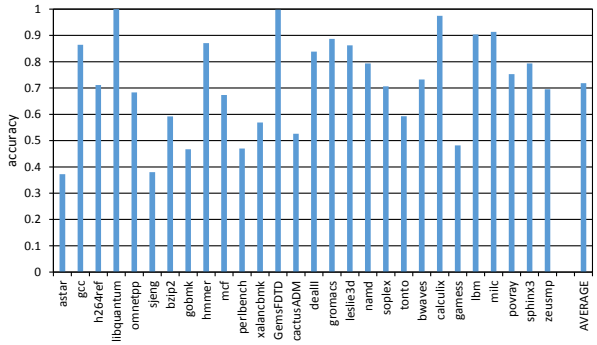


図 5　POPT 置換アルゴリズムの予測精度

ミスを減らし，ミス率を大きく削減することができる．

4. 評価

4.1 評価環境

評価には，メモリ・アクセス・トレースを入力とするキャッシュ・シミュレータ作成し，使用した．メモリ・アクセス・トレースの取得には，SimpleScalar Tool Set Version 3.0a[16] を用いた．命令セットは Alpha ISA を使用した．ベンチマーク・プログラムとして，SPECint2006 の全 12 本，SPECfp2006 から wrf を除く 16 本を使用した．プログラムは，gcc ver. 4.5.3 でコンパイルし，コンパイル・オプション-O3 を用いた．SimpleScalar の入力には ref データ・セットを用い，先頭の 16G 命令をスキップした後の 100M 命令で測定を行った．FC の設定は 256B，32B ライン，完全連想として測定した．これは，FC 中にキャッシュ・ラインが 8 本あることを意味する．

4.2 オーバーヘッドの見積もり

本節では提案手法に必要なオーバーヘッドの容量について見積もる．

(1) NAPT
NAPT は 2K エントリ，各エントリは 7 ビット (有効ビット 1 ビット，記録するアクセス間隔 6 ビット) とした．したがって，NAPT の容量は 2K×7 ビット =1.792KB である．

(2) VB
VB は 4 エントリ，各エントリは 78 ビットとした．各エントリに必要な 78 ビットの内訳は，タグとしてライン・アドレス (59 ビット)，DATC(6 ビット)，NIF(11 ビット)，VB のエントリを LRU で管理するための置換優先度 (2 ビット) である．したがって，VB の容量は 4×78 ビット = 39B である．

(3) FC の追加容量
FC は 8 ラインあり，各ラインに追加するビットは 24 ビットである．この 24 ビットの内訳は，DATC(6 ビット)，NIF(11 ビット)，CDT(7 ビット)，PVB(1 ビット) である．CDT は次回アクセスまでの間隔をカウントダウンし，0 になった後もしばらくデクリメントし続けるために，間隔を 1 ビット多く保持できるようにしている．したがって，FC に追加する容量は 8 ライン ×25 ビット = 25B である．

以上より，提案手法はおよそ 1.8KB のオーバーヘッドで構成することができる．VB と FC に追加する容量は非常に小さく構成でき，NAPT のオーバーヘッドが最も大きい．1.8KB の容量は，FC よりは大きいが，L1 D キャッシュなどよりは非常に小さく，大きな電力増加はないと考える．

4.3 予測精度

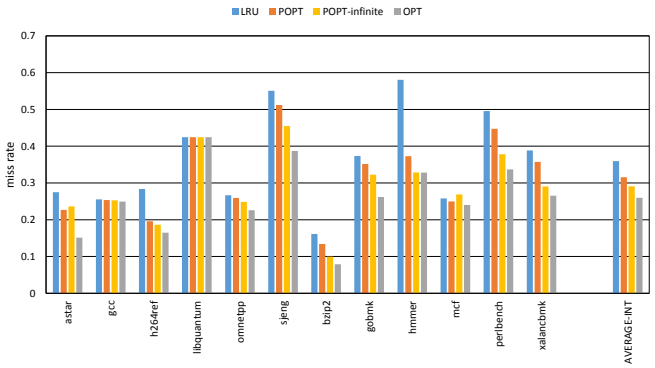
本節では，提案手法の予測精度について評価する．測定結果を**図 5** に示す．図の横軸はベンチマーク，縦軸は予測精度を示す．予測精度は，全キャッシュ・アクセスで次回のアクセスのタイミングを予測し，この予測ができた中で予測が正しかった割合とする．予測が正しいとは，予測した次回アクセスのタイミングちょうどにアクセスが来た場合と定義した．精度は全ベンチマークの平均で，71.8%を達成している．libuquantum，GemsFDTD，calculix, lbm, milc では，精度がそれぞれ 100.0%，99.8%，97.5%，90.4%，91.4%と 9 割以上の予測精度を達成した．一方で，astar，sjeng，gobmk，perlbench，gamess では，精度がそれぞれ 37.3%，38.0%，46.7%，47.0%，48.2%と他のベンチマークに比べて低くなっている．実際には，4.4 節で述べるように，予測したタイミングと多少ずれていても置換アルゴリズムはうまく働く場合が多い．

4.4 ミス率

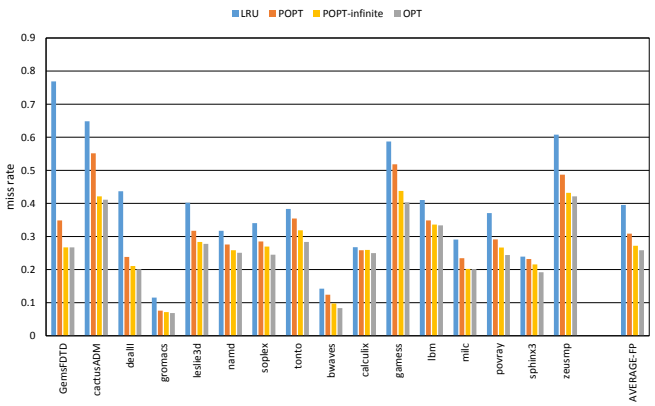
本節では提案手法を適用した場合の FC のミス率について評価する．ミス率の測定結果を**図 6** に示す．図の横軸はベンチマーク，縦軸はミス率を表す．各棒グラフは LRU 置換アルゴリズム，POPT 置換アルゴリズム，資源を無限大とした場合の POPT 置換アルゴリズム，OPT 置換アルゴリズムに対応しており，凡例を図上部に示す．

まず，LRU 置換アルゴリズムと比較をする．POPT 置換アルゴリズムは全ベンチマークで LRU 置換アルゴリズムからミス率を改善できている．最大で GemsFDTD で

Vol.2016-ARC-219 No.13
Vol.2016-SLDM-175 No.13
Vol.2016-EMB-40 No.13
2016/3/24



(a) SPECint2006



(b) SPECfp2006

図 6 キャッシュ・ミス率

76.9%から 34.9%へ半分以上ミス率を削減した．全ベンチマークの平均では，38.0%から 31.2%へ 17.9%のミス率の改善を達成できている．

次に，OPT 置換アルゴリズムと比較をする．OPT 置換アルゴリズムは最もミス率が低くなる理想的なアルゴリズムであるため，これより低いミス率を達成しているベンチマークはない．全ベンチマークの平均では，OPT 置換アルゴリズムの 25.9%に対し，POPT 置換アルゴリズムでは 31.2%と 20%差のミス率まで達成することができている．前節の評価で予測精度の低かった astar，sjeng，gobmk，perlbench，gamess では，OPT 置換アルゴリズムとの差が比較的大きい．

もし，より多くの資源を使うことを許容できれば，POPT-infinite に示すように，最大で 28.0%のミス率を達成することができる．

5. まとめ

本論文では，フィルタ・キャッシュのミス率を改善するために POPT 置換アルゴリズムを提案した．およそ 1.8KB の資源を仮定して評価した結果，提案手法によりフィルタ・キャッシュのミスを 17.9%削減し，31.2%のキャッシュ・ミス率を達成した．また，理想的な OPT 置換アルゴリズムのミス数 20%差まで近づくことができた．

謝辞 本研究の一部は，日本学術振興会 科学研究費補助金 基盤研究 (C)（課題番号 25330057），および日本学術振興会 科学研究費補助金 若手研究 (A)（課題番号 24680005）による補助のもとで行われた．

参考文献

[1] Gwennap, L.: Cortex-A35 Extends Low End, *Microprocessor Report* (2015).
[2] Demler, M.: Cortex-M7 Doubles Up on DSP, *Microprocessor Report* (2014).
[3] Krewell, K.: Cortex-A53 Is ARM's Next Little Thing, *Microprocessor Report* (2011).
[4] Kin, J., Gupta, M. and M.-Smith, W. H.: The Filter Cache: An Energy Efficient Memory Structure, *Proceesings of the 30th Annual International Symposium on Microarchitecture*, pp. 184–193 (1997).
[5] Tanenbaum, A. S.: *Modern Operating Systems*, Prentice Hall Press, 2nd edition (2001).
[6] Belady, L.: A study of replacement algorithms for a virtual-storage computer, *IBM Systems Journal*, Vol. 5, No. 2, pp. 78–101 (1966).
[7] Lin, W.-F. and Reinhardt, S.: Predicting Last-Touch References under Optimal Replacement, Technical Report CSE-TR-447-02, University of Michigan (2002).
[8] Lai, A.-C., Fide, C. and Falsafi, B.: Dead-block Prediction & Dead-block Correlating Prefetchers, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 144–154 (2001).
[9] Kaxiras, S., Hu, Z. and Martonosi, M.: Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 240–251 (2001).
[10] Abella, J., González, A., Vera, X. and O'Boyle, M. F. P.: IATAC: A Smart Predictor to Turn-off L2 Cache Lines, *ACM Transactions on Architecture and Code Optimization*, pp. 55–77 (2005).
[11] Kharbutli, M. and Solihin, Y.: Counter-Based Cache Replacement and Bypassing Algorithms, *IEEE Transactions on Computers*, Vol. 57, No. 4, pp. 433–447 (2008).
[12] Liu, H., Ferdman, M., Huh, J. and Burger, D.: Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency, *Proceedings of the 41st Annual International Symposium on Microarchitecture*, pp. 222–233 (2008).
[13] Qureshi, M. K., Jaleel, A., Patt, Y. N., Steely, S. C. and Emer, J.: Adaptive Insertion Policies for High Performance Caching, *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pp. 381–391 (2007).
[14] Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Steely, Jr., S. and Emer, J.: Adaptive Insertion Policies for Managing Shared Caches, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 208–219 (2008).
[15] Jaleel, A., Theobald, K. B., Steely, Jr., S. C. and Emer, J.: High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP), *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pp. 60–71 (2010).
[16] *http://www.simplescalar.com/*.