Vol.2010-ARC-190 No.13
2010/8/3

マップ型履歴を用いたプリフェッチ方式とキャッシュ置換方式の協調動作

石井康雄[†1,†2] 稲葉真理[†1] 平木敬[†1]

本論文では，マップ型履歴を採用するプリフェッチ方式の情報を利用してキャッシュ置き換えを行う Cache Replacement based on Access map Pattern matching (CRAP) を提案する．CRAP はキャッシュメモリ上で将来に渡って利用されないキャッシュライン (Dead-Block) を予測する．CRAP はプリフェッチ方式がサポートするメモリアクセスパターンが時間的局所性に乏しい，という性質を利用して Dead-Block 予測を行う．Dead-Block を他のキャッシュラインよりも優先的に追い出すことでキャッシュメモリの利用効率を改善する．CRAP をキャッシュシミュレータを用いて評価した．その結果，CRAP は LRU からキャッシュミスを 2.0%削減することが分かった．

Interaction between Cache Replamcement Policy and Hardware Data Prefetching using Map-based History

Yasuo Ishii,[†1,†2] Mary Inaba[†1] and Kei Hiraki[†1]

In this paper, we propose Cache Replacement based on Access map Pattern matching (CRAP). CRAP predicts cache lines that are never reused (Dead-Blocks). CRAP predicts Dead-Block based on the prefetching results because CRAP assumes the memory access pattern detected by the data prefetching has insufficient temporal locality. CRAP improves the cache efficiency by evicting the Dead-Blocks earlier than other cache lines. We evaluate CRAP by the simulation. The result shows that CRAP reduces cache miss counts by 2.0% from LRU policy.

†1 東京大学
The University of Tokyo
†2 日本電気株式会社
NEC Corporation

1. はじめに

近年の半導体プロセスの進化はプロセッサの処理速度を飛躍的に向上させてきた．しかし，メモリアクセスの速度向上はプロセッサの速度向上に比べ小さく，主記憶へのアクセスレイテンシは相対的に増加し，メモリバンド幅は減少する傾向にある．主記憶へアクセスしたデータを待つことによって発生するデータハザードはプロセッサの効率的な実行を阻害する．この問題はメモリウォール問題として知られている．メモリウォール問題は現代のプロセッサにおける最も重要な課題の 1 つであり，その緩和のために様々な手法が提案されてきた．

キャッシュメモリは頻繁にアクセスするメモリデータを高速な RAM に保持することでデータを読み出す際の平均のメモリレイテンシを削減し，無駄な主記憶アクセスを削減する．キャッシュメモリの効率を最大限に発揮するためには将来利用されるデータを保持することが重要である．一方，キャッシュ置き換え方式はキャッシュメモリ上に配置するデータの選択方式である．キャッシュメモリに保持しているデータから，将来利用される可能性が高いデータを予測してキャッシュ残すことが求められる．キャッシュ置き換え方式では，もっとも長く参照されないデータを追い出す OPT 方式[2] がキャッシュメモリの利用効率を最大化することが知られている．しかし，OPT 方式は未来のメモリアクセスの情報が必要であり，実現は不可能である．OPT 方式に代わる方法として，多くの計算機で LRU 方式が利用されてきた．LRU 方式では最も最近アクセスされたデータを LRU スタックの Most Recently Used(MRU) 側に挿入し，LRU スタックの Least Recently Used(LRU) 側のキャッシュラインを追い出す．これは recency に基づく置き換え方式であり，ワークロードサイズがキャッシュメモリの容量よりも小さい場合や，スタックモデルに基づくアクセスパターンが多い場合は効率的に動作する．しかし，LRU 方式ではワークロードサイズがキャッシュメモリと比べ巨大で，ストリームアクセスなどスタックモデルに基づかないメモリアクセスパターンを採用するアプリケーションでは，キャッシュのスラッシングが発生し，効率的に動作しないことが知られている[11]．

このようなワークロードを持つアプリケーションに対応するため，近い将来キャッシュメモリ上で利用される可能性が低いデータを積極的に追い出す方式が提案されてきた．これらの方式を本論文では Dead-Block 予測と呼ぶ．Dead-Block 予測を行うキャッシュ置き換え方式は過去のメモリアクセス履歴から既にキャッシュに存在するデータの中で将来にわたって利用されないデータを予測する．キャッシュメモリの置き換え時には，上記の予測結果に

© 2010 Information Processing Society of Japan

基づき，有用であると予測されるデータをキャッシュメモリに残し，不要であると予測されるデータをキャッシュメモリから追い出す．Dead-Block 予測が正確であれば，不要であると予測されるデータをキャッシュメモリから追い出すことでキャッシュメモリの利用効率が改善され，性能が向上する．

キャッシュメモリの性能を更に向上させる方法として，ハードウェアデータプリフェッチが有効である．ハードウェアデータプリフェッチ[14] は将来のメモリアクセスパターンを予測し，実際のメモリアクセスが発生する前にアクセスされると予想したデータをキャッシュメモリに読み出し，メモリアクセスのレイテンシを削減する方式である．キャッシュ置き換え方式と異なる点は現在キャッシュメモリにないデータで将来利用される可能性が高いものを予測する点である．様々なメモリアクセスパターンを予測する方式が提案されてきたが，その多くは連続メモリ領域へのアクセス[12] やストライドアクセス[1] を予測する．このようなハードウェアプリフェッチ方式は既に一部のプロセッサで利用されている．これらのメモリアクセスパターンは巨大な配列などの膨大なデータに対するアクセスを行うため，メモリアクセスの時間的局所性に乏しい．このようなワークロードではキャッシュメモリは効率的に動作しないため，プリフェッチによる性能向上が有効である．

Dead-Block 予測を行うキャッシュ置き換え方式とプリフェッチはともに巨大なワークロードに対して有効であることがわかる．プリフェッチ方式が予測対象とするストライドやシーケンシャルアクセスパターンはデータの再利用性に乏しく，これらのメモリアクセスパターンの情報は Dead-Block 予測に対しても有益である．

将来のメモリアクセスの予測をマップ型履歴を用いて実現するプリフェッチ方式[4] やキャッシュ置き換え方式[5] が最近になって著者らにより提案された．これらの方式は固定サイズのメモリ領域のメモリアクセス履歴をビットマップ型のデータ構造に保持する．このデータ構造からはメモリアクセスパターンやメモリアクセスの局所性を効率的に取り出すことが可能で，高いコスト効率が得られる．

本論文では，プリフェッチが予測したメモリアクセスパターン情報を利用したキャッシュ置換方式 Cache Replacement based on Access map Pattern matching(CRAP) を提案する．この方式ではマップ型履歴を利用したプリフェッチ方式の Access Map Pattern Matching (AMPM) の予測結果から将来利用される可能性の低いラインを予測する Dead-Block 予測を用いたキャッシュ置き換え方式である．プリフェッチが用いるメモリアクセスパターン情報をキャッシュメモリの置き換え方式にも利用することでハードウェアを効率的に利用しつつ，高い Dead-Block 予測精度を実現する．

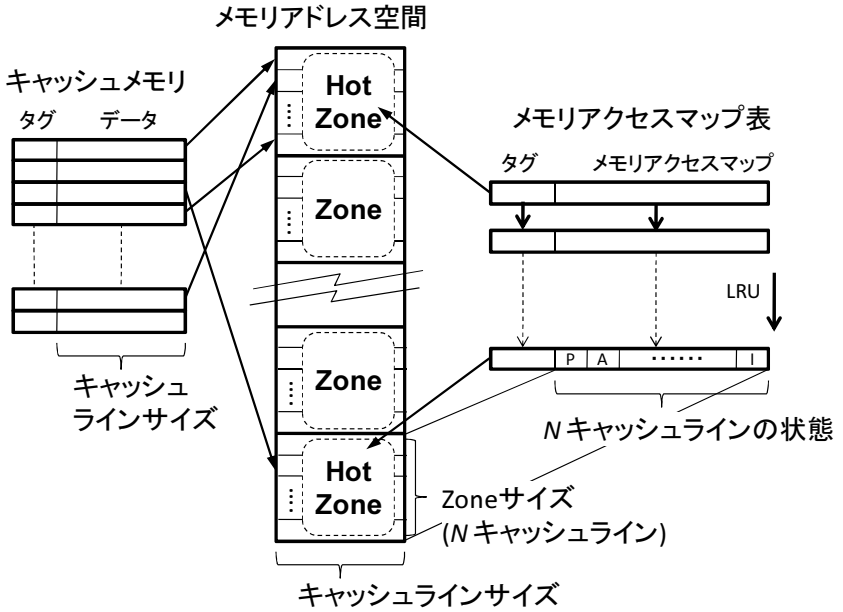

図 1 メモリアクセスマップのイメージ図
Fig. 1 Overview of Memory Access Map

本論文の構成は，以下の通りである．2 章でマップ型履歴とその既存手法に関して述べる．3 章で CRAP を提案する．4 章でキャッシュシミュレータを用いて評価を行う．5 章で関連研究に関して述べ，6 章でまとめる．

2. メモリアクセスマップ

メモリアクセスマップは連続するメモリ領域のメモリアクセス履歴を記録するデータ構造である．ビットマップ型のデータ構造をとることにより効率良くメモリアクセス履歴を記録することが出来る．**図 1** にメモリアクセスマップの概要を示す．メモリアクセスマップはゾーンとよばれる固定領域のメモリアドレス空間に対してマッピングされる．メモリアクセスマップはビットマップ型のデータ構造でメモリアクセス履歴を記憶する．マップ中の各ステートはマッピングされたゾーンのキャッシュラインに対してマップされる．各ステートには関連付けられたキャッシュラインのメモリアクセス履歴を記録する．ゾーン単位でマッピ

© 2010 Information Processing Society of Japan

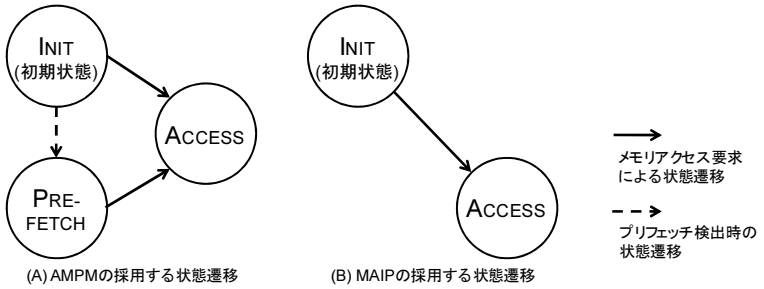

図 2 アクセスマップで利用する状態遷移図
Fig. 2 State Diagrams of Memory Access Map

ングされるため，各ステートはマッピングされたゾーンの場所を示すタグを保持する必要がない．シャドウタグ[15] などのデータ構造と比較して，1 つのゾーンに対してアクセスが集中する場合には高効率を実現する．

以下では，プリフェッチ方式の応用例である AMPM プリフェッチ方式と，キャッシュ置き換え方式の応用例である MAIP に関して説明する．

2.1 Access Map Pattern Matching(AMPM)

Access Map Pattern Matching (AMPM) は[4] Data Prefetching Championship (DPC-1) で著者らが提案した高性能なプリフェッチ方式である。AMPM では各キャッシュラインに Init, Access, Prefetch の 3 状態を与える．この状態遷移を**図 2** に示す．状態遷移はキャッシュメモリのアクセスとプリフェッチ要求によって行われる．

AMPM は順序情報や命令アドレス情報のないメモリアクセスマップからパターン検出でメモリアクセスのパターンを抽出し，プリフェッチを実施する．パターン検出はメモリアクセス命令によって生じたメモリアクセス要求をトリガに実施される．AMPM はメモリアクセスマップに接続されているパターンマッチング回路により，複数のストライドアクセスのパターン検出を並列して行う．AMPM は検出したパターンに基づきプリフェッチ要求を生成する．なお，AMPM が検出するストライドアクセスのパターンは時間的局所性に乏しいため，単純にキャッシュメモリの容量を増やすなどのアプローチによる性能改善は難しい．

2.2 Map-based Adaptive Insertion Policy(MAIP)

Map-based Adaptive Insertion Policy(MAIP)[5] は Cache Replacement Championship (JWAC-1) で著者らが提案した高性能なキャッシュ置き換え方式である。MAIP では各キャッシュラインに Init, Access の 2 状態を与え，状態遷移からそのゾーンへのアクセスがどのくらい再利用されるかを予測する．

MAIP は 2 つのカウンタ (Access カウンタと Reuse カウンタ) をメモリアクセスマップに対して付与し，Dead-Block の予測に用いる．Access カウンタは Init 状態へのメモリアクセスの回数をカウントし，Reuse カウンタは Access 状態へのメモリアクセスの回数をカウントする．一度アクセスされたステートは Access 状態へ遷移するため，メモリアクセスの再利用が多いゾーンでは Reuse カウンタの値が大きくなり，メモリアクセスの再利用が少ないゾーンでは Access カウンタの値が大きくなる．さらに，この情報は各メモリアクセスマップに付与されるため，Access カウンタと Reuse カウンタの値は周辺のメモリアクセスの状況を反映したものとなる．

MAIP はキャッシュ置き換えの際に周辺のメモリアクセスマップの Access カウンタと Reuse カウンタの値を比較してキャッシュメモリへの LRU スタックの挿入位置を変更する．Reuse カウンタの値が Access カウンタよりも大きい場合に再利用性が高いと判断し，Reuse カウンタの値が Access カウンタよりも小さい場合に再利用性が低いと判断する．ただし，新規にアクセスしたゾーンでは Access カウンタ，Reuse カウンタ共に 0 でクリアされるため，各ゾーン単位でトレーニング期間が必要となり，全てのキャッシュラインに対する Dead-Block 予測を行うことができない．

3. Cache Replacement based on Access map Pattern matching (CRAP)

我々は AMPM と MAIP が似たデータ構造を用いて，データプリフェッチとキャッシュ置き換え方式の 2 つの目的に活用できることに着目し，AMPM プリフェッチ方式が検出したストライドアクセスパターンの情報をキャッシュ置き換え方式に利用する Cache Replacement based on Access map Pattern matching (CRAP) を提案する．CRAP は AMPM の予測したプリフェッチ先のアドレスの再利用可能性が低いことに着目し，その情報を用いて Dead-Block 予測を行うキャッシュ置き換え方式である．CRAP はプリフェッチに成功したデータは時間的局所性が乏しいメモリアクセスパターンであると判断し，そのデータを積極的に追い出す．

MAIP もメモリアクセスマップを用いるキャッシュ置き換え方式であるが，Dead-Block 予測を行うタイミングが異なる．MAIP はキャッシュへのデータ挿入時点で Dead-Block 予測を行い，LRU スタックへの挿入位置を変更することで不要なラインの早期の追い出しを

© 2010 Information Processing Society of Japan

実現する．一方，CRAP はキャッシュへデータが挿入されたのちにプリフェッチの情報を用いて Dead-Block 予測を行う．MAIP は既存のキャッシュの構造に対して大幅な変更をせずに実現できるが，キャッシュ挿入時点でそのブロックの有用性を予測する必要がある．対して，CRAP はキャッシュメモリにフラグを追加する必要があるが，データのアクセスの結果を用いて Dead-Block 予測ができるため，利用できる情報量が多いという利点がある．

3.1 概　　要

CRAP は AMPM の状態遷移からプリフェッチ先のアドレスとそのプリフェッチの成功と失敗を推論する．ここで，プリフェッチの成功とは，プリフェッチしたキャッシュラインが実際にアクセスされたことをいう．プリフェッチの失敗とは，プリフェッチしたキャッシュラインが実際にアクセスされる前にキャッシュから追い出されたことをいう．図 2 に示した通り，AMPM ではプリフェッチしたキャッシュラインと関連付けられるメモリアクセスマップ上のステートは Prefetch 状態へと遷移する．そのキャッシュラインに対してアクセスがあると Prefetch 状態は Access 状態へと遷移するため，プリフェッチが成功した，と推論することができる．前節で述べたとおり，データプリフェッチ方式はデータの時間的局所性に乏しい連続領域へのアクセスやストライドアクセスパターンを予測する．プリフェッチに成功したキャッシュラインはこのメモリアクセスパターン中でアクセスされており，時間的局所性に乏しいと予測することができる．

CRAP では，時間的局所性の予測結果を保持するために，Dead フラグという情報を各キャッシュラインに対して付加する．プリフェッチの成功を確認したラインには Dead フラグをセットして，そのキャッシュラインの時間的局所性が乏しいことを記録する．キャッシュミスが発生した際には，Dead フラグのセットされたキャッシュラインが最優先で追い出すことで，将来利用される可能性の高いキャッシュラインの保護が実現できる．なお，Dead フラグがセットされたラインに対して，通常のメモリアクセスが発生した場合には，Dead-Block 予測が失敗したと判断して，Dead フラグをリセットする．

3.2 Dead-Block 予測を用いたキャッシュ置き換え

CRAP は Dead-Block 予測を用いて，不要なキャッシュラインを積極的に追い出すキャッシュ置き換えを実現する．AMPM プリフェッチ方式の出力するプリフェッチの成功情報を用いて，そのキャッシュラインの時間的局所性が乏しいことを検出する．検出した Dead-Block 予測結果は，予測対象のキャッシュラインに付与される 1 bit の情報 (Dead フラグ) に保持する．キャッシュ置き換え時点では既存の LRU ラインの追い出しよりも Dead フラグ付きのラインの追い出しを行うことで効率的な Dead-Block 予測を実現する．この予測機構は

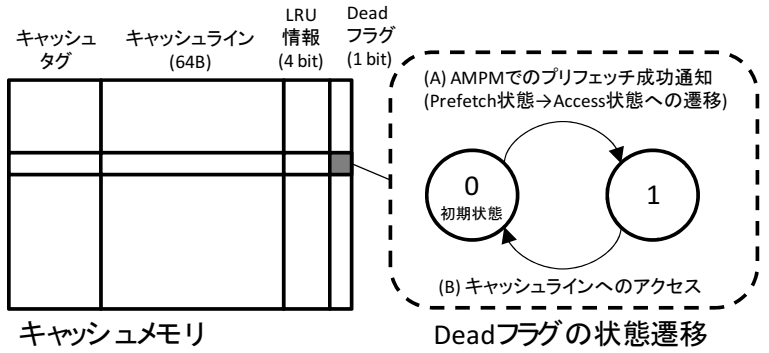

図 3　Cache Replacement based on Access map Pattern matching の概要
Fig. 3　Cache Replacement based on Access map Pattern matching

既存のプリフェッチ方式である AMPM によるプリフェッチ部分と新規に追加されるキャッシュラインへ追加するフラグで構成できる．Dead-Block 予測のためのハードウェアの構成を**図 3** に示す．

CRAP は各キャッシュラインに対して LRU 情報と Dead フラグを付加する．LRU 情報は N-way のセットアソシアティブキャッシュで $log_2 N$ ビットの情報を付加する．LRU 情報は広く利用されている LRU 置き換え方式と同じ情報である．Dead フラグはそのキャッシュラインが Dead-Block として予測されているかを示す 1 bit のフラグである．関連するキャッシュラインへのプリフェッチ成功でセットし，キャッシュヒットでリセットする．CRAP は Dead フラグが点灯しているキャッシュラインは再利用さる可能性が低いと判断し積極的に追い出す．

以下では，動作の詳細に関して説明する．コアからメモリアクセス要求が発生すると，AMPM とキャッシュメモリの両方に対してメモリアクセス要求が発行される．AMPM では要求に基づいてメモリアクセスパターン検出を行い，プリフェッチ要求を発生させる．その際に，自身のメモリアクセスマップの更新を行う．もしも，AMPM が関連するメモリアクセスで Prefetch 状態から Access 状態への遷移を検出した場合には，そのことをキャッシュメモリに対して通知する．この通知を受け取ると，キャッシュメモリでは関連するキャッシュラインに対して Dead フラグをセットする．

メモリアクセス要求は AMPM へのプリフェッチ情報の更新と並行してキャッシュメモリに対しても発行される．キャッシュメモリはメモリアクセス要求のアドレスに基づいてヒッ

© 2010 Information Processing Society of Japan

ト/ミス判定を実施する．キャッシュヒットだった場合には，LRU 情報を MRU 側に更新し．Dead フラグがセットされていた場合はリセットする．キャッシュミスだった場合には，該当する N-way のセットから追い出すキャッシュラインを選択する．この追い出しラインの選択は以下の優先度で行う．(1) Dead フラグがセットされているキャッシュライン，かつ，最も LRU 側に位置するキャッシュラインを追い出す．(2) 全てのキャッシュラインの Dead フラグがセットされていない場合，LRU のキャッシュラインを追い出す．その後，メモリから取得した新しいキャッシュラインを MRU に挿入し，Dead フラグをリセットする．もしも，AMPM がそのキャッシュラインで Prefetch 状態から Access 状態への遷移を検出していた場合には，Dead フラグは強制的にセットされる．これは前述の Dead フラグのリセット操作よりも優先して実施される．すなわち，プリフェッチ成功通知があったメモリアクセスでは最終的に Dead フラグが必ずセットされる．

3.3 Set Dueling を用いたポリシー選択

CRAP は Dead フラグのセットされたキャッシュラインが再利用されないものとして設計されている．しかし，この前提が必ずしも全てのワークロードに当てはまるとは限らない．例えば，プリフェッチしたデータが 2 回アクセスされると Dead-Block 予測は外れることとなる．もしも，そのようなワークロードを実行する際には CRAP は Dead-Block 予測による積極的な追い出しを中止して，LRU 置き換えに従ったキャッシュ置き換えを実施するべきである．これを実現するために CRAP では LRU と Dead-Block 予測を用いたキャッシュ置き換えのどちらの方式が良いかを Set Dueling 方式[11] によって選択する．

Set Dueling 方式は DIP, MAIP などの多くのキャッシュ置き換え方式で利用されるポリシー選択方式である．Set Dueling 方式ではキャッシュセット中でいくつかの代表的なセット (Dedicated Set) を定めて，残りのキャッシュライン (Follower Set) と区別する．Dedicated Set の半分は常に LRU 方式でキャッシュ置き換えを行い，残りの半分は常に Dead-Block 予測を利用するキャッシュ置き換えを行う．Set Deuling 方式では 2 つのポリシーのどちらが有効であるかを表すカウンタを持つ．このカウンタを Policy Selection Counter (PSEL) と呼ぶ．LRU の Dedicated Set でキャッシュミスが発生すると PSEL を +1 し，Dead-Block 予測に基づくキャッシュ置き換え方式を採用する Dedicated Set でキャッシュミスが発生すると PSEL を −1 する．Follower Set は PSEL の値が一定値以上であれば Dead-Block 予測を用いたキャッシュ置き換えを行い，そうでない場合には LRU で置き換えを実施する．

Set Dueling 方式を採用することで最小限のハードウェア量の追加で大きな性能向上を得ることができる．Set Dueling をさらにマルチスレッド対応に拡張した方式[6] の採用も考えられるが，それは今後の課題とする．

表 1 シミュレータのパラメータ
Table 1 Simulation Parameter

	容量, 連想度，置き換え方式
L1 Data Cache	32KB, 4-way セットアソシアティブ, LRU
L1 Instruction Cache	32KB, 4-way セットアソシアティブ, LRU
L2 Unified Cache	256KB, 8-way セットアソシアティブ, LRU
L3 Unified Cache	2MB, 16-way セットアソシアティブ, CRAP
Prefetcher	AMPM Prefetch 256 state / map, 256 map, 16-way

4. 評　　価

4.1 評 価 環 境

評価は pin ツール[9] を使用して開発したキャッシュシミュレータを用いる．各ベンチマークははじめの 40G 命令をスキップして，次の 1G 命令のキャッシュミス回数を計測した．評価に利用したキャッシュ構成を**表 1** に示す．命令とデータに分離された L1 キャッシュに加え，L2 キャッシュと L3 キャッシュを採用する．L1 キャッシュ，L2 キャッシュでは LRU での置き換えを実施して，CRAP は L3 キャッシュの置き換え方式として利用する．

CRAP を実現するためには，AMPM プリフェッチ方式が必要となる．今回の評価では表 1 に示すパラメータを用いた．1 状態に 2 ビット必要であるため 16KB 程度のハードウェア資源が必要である．

また，比較対照として LRU，DIP[11] を評価に用いる．DIP は BIP ポリシーの LRU 挿入の選択パラメータとして $\epsilon = 1/32$ を用いた．

4.2 評 価 結 果

図 4 に 1000 命令あたりのキャッシュミス回数を示す．CRAP は 3.0 MPKI を超えるベンチマークで LRU と比較すると，平均して 2.0%のキャッシュミスを削減したことが確認できた．482.sphinx3 で，最もキャッシュミスの削減量が大きく，14.8%のキャッシュミスを削減した．一方，DIP は 3.0 MPKI を超えるベンチマークで平均して 10.2%のキャッシュミスを削減し，3.0 MPKI を超える全てのベンチマークで CRAP を上回る結果となった．

我々は，CRAP と DIP の Dead-Block の予測能力の差を確認する為に、各方式の予測精度とカバレッジを計測した．その結果を**図 5** に示す．図中で，**OVER** は Dead-Block 判定されたが最終的に利用されたデータの割合 (Dead-Block 予測失敗)，**COVER** 正しく判

© 2010 Information Processing Society of Japan

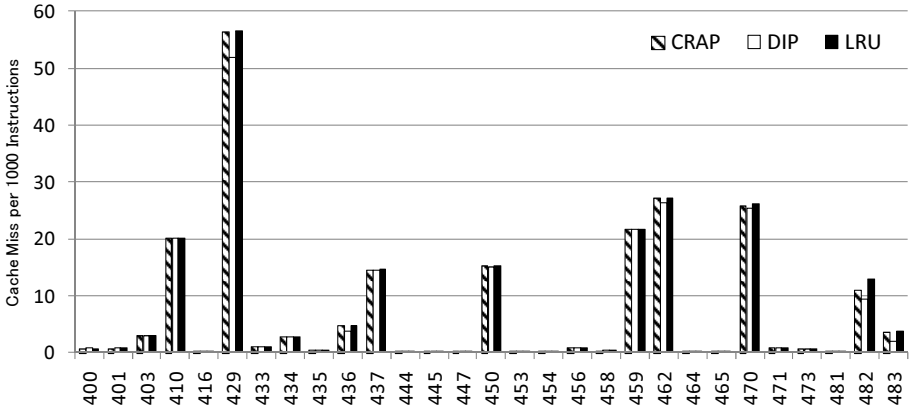

図 4 L3 キャッシュメモリでのキャッシュミス回数
Fig. 4 Cache miss counts on L3 cache memory

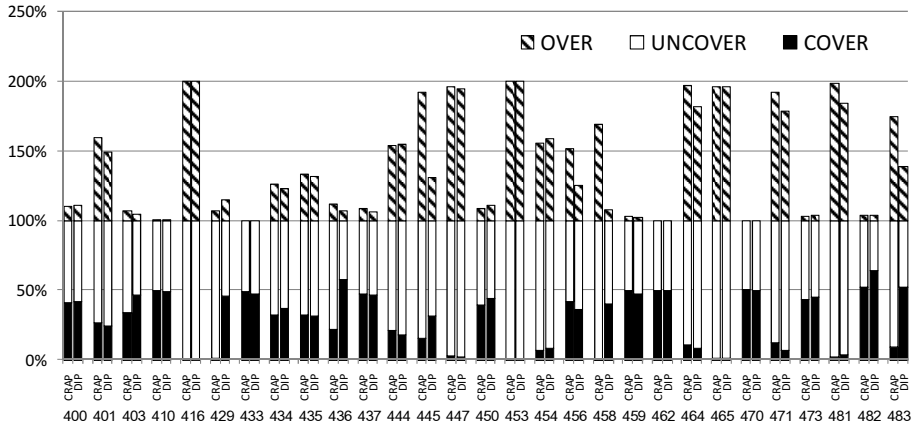

図 5 CRAP の Dead-Block 予測の成功率
Fig. 5 Accuracy of Dead-Block prediction by CRAP

定した割合 (Dead-Block 予測成功)，**UNCOVER** は Dead-Block であったが検出することのできなかったものを表す．458.sjeng, 483.xalan のベンチマークでは，提案手法がカバレッジが低く，かつ，予測精度も低いという結果が確認できた．これは，AMPM プリフェッチ方式自身の予測が正確でないために発生しているものと予想される．その他のベンチマークでは，平均すると予測精度，カバレッジとも 5%以内の差であり，方式間での大きな差はないと考えられる．

5. 関 連 研 究

5.1 LRU を用いないキャッシュ置換方式

LRU は古来から利用されれている方式であるが，セットアソシアティブの Way 数が増加すると，LRU 情報の再計算が煩雑になるため，スケーラビリティに欠ける方式である．また，マルチコアの共有キャッシュではコアの占有するデータの量が参照回数に比例してしまうため，不公平を招きやすいという問題も持つ．これらの問題を解決するため，種々の置換方式が提案されてきた．

Janapastya らは低コストなページ置換アルゴリズムとして知られる，CLOCK をキャッシュ置換ポリシーのために拡張した Dueling CLOCK 方式を提案した[8]．この方式では CLOCK による利用ビットの消去の巡回を止めることでスキャンのようなメモリアクセスに強いキャッシュ置き換えを実現する．

Jaleel らは広く利用される低コストな置換方式である Not Recently Used(NRU) 方式を拡張した RRIP を提案した[7]．この方式は LRU よりも低コストでスキャンやキャッシュスラッシングに強く，高い性能を実現する．

これらの方式は CRAP と平行して動作させることが可能である．なぜならば，CRAP は既存の LRU の置き換えアルゴリズム上に自然な形で Dead-Block 予測による不要なキャッシュラインの追い出しを実装しているためである．

5.2 Dead-Block 予測を行うキャッシュ置換方式

Lai らは過去の履歴から将来参照される可能性の低いブロックを検出し，そのデータをキャッシュメモリから積極的に追い出す Dead-Block 予測と Dead-Block の検出をトリガにプリフェッチを行う Dead-Block Correlating Prefetcher を提案した[10]．この Dead-Block 予測では，実行パス上の命令アドレスの値と，参照されるキャッシュブロックの相関から L1 キャッシュに含まれるデータの有用性を予測している．しかし，現代の Out-of-order 実行やマルチスレッド実行，中規模で高速な Middle Level Cache(MLC) を持つプロセッサにおいては L1 キャッシュのヒット率が性能に与える影響は小さく，性能の改善幅は限定的である．Dead-Block Correlating Prefetcher は本研究とは異なり，キャッシュ中で不要なブロックを検出した際に，それをより有用なデータで置き換えるという目的で行われる．しか

© 2010 Information Processing Society of Japan

し，メモリウォール問題の厳しい現代のプロセッサでは，無理に送出したプリフェッチ要求がメモリバス帯域を占有する問題の影響が大きい．

Xiang らは頻繁にアクセスされないデータを L2 キャッシュに挿入せずに，平行して配置したバッファに対して挿入する Less Reuse Filter(LRF) を提案した[15]．LRF は Victim キャッシュのように動作するバッファを追加のキャッシュとして利用する．Less Reuse Filter は各キャッシュラインの残りの参照回数をメモリアクセスパターンから予測する．残りの参照回数が 0 と予測されたものをバッファから追い出すことで，バッファ内部に参照可能性の高いデータを保持することを実現した．LRF は高い性能を実現するが，メモリアクセスパターンから残りの参照回数を記録するテーブル (PHT) や，キャッシュラインのタグ情報を保持するシャドウダグなどの多くの追加機能を必要とし，コスト性能面で CRAP を含む他方式に劣る．

5.3 空間的局所性を利用したプリフェッチ方式

プリフェッチ方式には，アクセスしたデータがキャッシュから追い出された後に再びアクセスされることを予測する時間的局所性を利用するものと，連続領域へのメモリアクセスやストライドなどのメモリアクセスパターンから予測する空間的局所性を利用するものがある．近年，プロセッサのキャッシュメモリの巨大化によって，時間的局所性があるデータがキャッシュメモリから追い出されるケースが稀になっており，空間的局所性を用いるプリフェッチ方式が注目されている．

Somogyi らが提案した Spatial Memory Streaming[13] はメモリアドレス空間を固定サイズで分割する．そのメモリ領域中で検出したメモリアクセス履歴をビットマップ化し，その領域に対して最初にメモリアクセスを発行した命令のアドレスとペアにして主記憶上に保存する．過去にメモリアクセスがなかった領域に対してメモリアクセスが発生すると，そのメモリアクセスを発生させた命令のアドレスをキーにして保存したメモリアクセスパターンを読み出しプリフェッチを行う．メモリアクセス履歴をビットマップ化する点が AMPM と似ているが，動的なパターン検出を用いる AMPM と比較して，予測が静的で保持するメモリアクセスパターンが膨大となり主記憶上に専用の領域を用いて管理するため，システム全体を複雑化するという問題点を持つ．

Hur らは連続するメモリアクセスを検出し，その長さを確率的手法を用いて予測する方式を提案した[3]．いくつかのアクセス数を記録するカウンタを用いてメモリアクセスの長さの予測を実現する．連続領域のメモリアクセス予測では少ないハードウェア量で効率的にメモリアクセスを予測するが，ストライドアクセスなどの複雑なパターンに基づく予測ができないため，性能の向上幅が限定的という欠点を持つ．

これらのプリフェッチ方式が予測したデータは時間的局所性に乏しい．従って，CRAP と同じく Dead-Block 予測に用いることも可能である．しかし，これらのプリフェッチ方式はプリフェッチの成功を推論する手段を持たないため，CRAP の採用した Dead-Block 予測を実現するには，キャッシュラインに対してさらなるビット追加が必要となる．

6. おわりに

本論文では，プリフェッチの検出したメモリアクセスパターン情報を利用して，キャッシュラインの将来のアクセスの有無を予測し，不要なキャッシュラインを積極的に置き換える Cache Replacement based on Access map Pattern matching(CRAP) を提案した．マップ型履歴を採用するプリフェッチ方式から情報を得ることで追加のハードウェアを削減しながら高い性能を実現する．また，CRAP の採用する AMPM プリフェッチ方式は高い性能を示すため，効率的にメモリアクセスの高速化を実現することができる．

CRAP 方式ではプリフェッチの成功するメモリアクセスパターンではデータの再利用が発生しにくい，という性質を利用することで再利用されないキャッシュラインを予測する (Dead-Block 予測)．CRAP は Dead-Block 予測をマップ型履歴を用いるプリフェッチ方式である AMPM とキャッシュラインに追加する 1 ビットのフラグ (Dead フラグ) で実現する．Dead-Block フラグのセットされたキャッシュラインをキャッシュ置き換えの際に優先的に追い出すことで，キャッシュメモリの利用効率を向上させる．また，Dead-Block 予測に基づく置き換えにマッチしないワークロードでの性能を確保するため，従来の LRU を動的に切り替えながら実行する．ポリシーの切り替えには Set Dueling を用いることで，追加のハードウェア量を 1 つのカウンタのみとし，効率的なハードウェア構成を実現している．

CRAP をキャッシュシミュレータによって評価した．その結果，CRAP は LRU からキャッシュミスを 2.0%削減することが分かった．この結果によってプリフェッチ情報によるキャッシュ置き換え性能の改善の可能性を示したが，DIP に代表される既存のキャッシュ置き換え方式に劣る性能であり，さらなる改善が必要であることがわかった．今後は，SRRIP などに代表される LRU よりも高い性能を実現するとされるキャッシュ置き換え方式に対して CRAP を評価し，プリフェッチからのメモリアクセスパターンを利用するキャッシュ置き換え方式の汎用性を評価すると共に，より小さなハードウェアコストでの Dead-Block 予測を実現していきたい．

© 2010 Information Processing Society of Japan

参 考 文 献

1) Baer, J.-L. and Chen, T.-F.: Effective Hardware-Based Data Prefetching for High-Performance Processors, *IEEE Trans. Comput.*, Vol.44, No.5, pp.609–623 (1995).
2) Belady, L.A.: A study of replacement algorithms for a virtual-storage computer, *IBM Syst. J.*, Vol.5, No.2, pp.78–101 (1966).
3) Hur, I. and Lin, C.: Memory Prefetching Using Adaptive Stream Detection, *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.397–408 (2006).
4) Ishii, Y., Inaba, M. and Hiraki, K.: Access Map Pattern Matching Prefetch: Optimization Friendly Method, *The first JILP Data Prefetching Championship (DPC-1)* (2009).
5) Ishii, Y., Inaba, M. and Hiraki, K.: Cache Replacement Policy using Map-based Adaptive Insertion, *First JILP Workshop on Computer Architecture Competitions (JWAC-1)* (2010).
6) Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Steely, Jr., S. and Emer, J.: Adaptive insertion policies for managing shared caches, *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY, USA, ACM, pp.208–219 (2008).
7) Jaleel, A., Theobald, K.B., Steely, Jr., S.C. and Emer, J.: High performance cache replacement using re-reference interval prediction (RRIP), *SIGARCH Comput. Archit. News*, Vol.38, No.3, pp.60–71 (2010).
8) Janapsatya, A., Ignjatovic, A., Peddersen, J. and Parameswaran, S.: Dueling CLOCK: Adaptive cache replacement policy based on the CLOCK algorithm, *DATE*, IEEE, pp.920–925 (2010).
9) keung Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Janapa, V. and Hazelwood, R.K.: Pin: Building customized program analysis tools with dynamic instrumentation, *In Programming Language Design and Implementation*, ACM Press, pp.190–200 (2005).
10) Lai, A.-C., Fide, C. and Falsafi, B.: Dead-block prediction & dead-block correlating prefetchers, *SIGARCH Comput. Archit. News*, Vol.29, No.2, pp.144–154 (2001).
11) Qureshi, M.K., Jaleel, A., Patt, Y.N., Steely, S.C. and Emer, J.: Adaptive insertion policies for high performance caching, *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, New York, NY, USA, ACM, pp.381–391 (2007).
12) Smith, A.J.: Sequential Program Prefetching in Memory Hierarchies, *Computer*, Vol.11, No.12, pp.7–21 (1978).
13) Somogyi, S., Wenisch, T.F., Ailamaki, A., Falsafi, B. and Moshovos, A.: Spatial Memory Streaming, *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pp.252–263 (2006).
14) Vanderwiel, S.P. and Lilja, D.J.: Data prefetch mechanisms, *ACM Comput. Surv.*, Vol.32, No.2, pp.174–199 (2000).
15) Xiang, L., Chen, T., Shi, Q. and Hu, W.: Less reused filter: improving l2 cache performance via filtering less reused lines, *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, New York, NY, USA, ACM, pp.68–79 (2009).

© 2010 Information Processing Society of Japan