

卒業研究論文

ロード命令およびエントリ毎の特徴に基づく 再参照間隔予測を用いたキャッシュ性能向上手法

指導教員 津邑 公暁 准教授
松尾 啓志 教授

名古屋工業大学 工学部 情報工学科
平成 21 年度入学 21115093 番

力 翠湖

平成 25 年 2 月 12 日

ロード命令およびエン트리毎の特徴に基づく 再参照間隔予測を用いたキャッシュ性能向上手法

力 翠湖

内容梗概

これまで、ゲート遅延に対する配線遅延の相対的増大により、これらの性能差を隠蔽するキャッシュメモリの重要性が高まってきた。キャッシュを高性能化させる様々な手法のひとつとして、キャッシュエントリに優先度を設けてリプレースに利用する手法が存在する。この手法では、エントリのリプレースの優先度を決定するために、キャッシュ上に配置されるエントリの特徴を抽出する。本研究では、それらのエントリの特徴に着目し、ロード命令毎に特徴を抽出する方式と、エントリ毎に特徴を抽出する方式に分類し、キャッシュリプレースの最適化について考察する。

この一方で、キャッシュリプレースアルゴリズムとして最適であると知られているOPTが存在する。OPTは再参照されるまでの期間が最も長いエントリをリプレースする方式である。しかし、この手法を実現するにはアクセスされるアドレスの順序情報を事前に知る必要があり、実装することは困難である。そこで、このOPTを擬似的に実現する手法としてShepherd Cacheが提案されている。Shepherd Cacheは、アクセスされるアドレスの順序情報の代わりにエントリ毎の特徴に基づく再参照間隔予測を用いてリプレース対象を決定する手法である。しかしながら、この手法は頻繁にヒットするエントリの再参照間隔は予測できるが、再参照されるまでの期間が長いエントリの予測はできない。そこで本研究では、エントリ毎の特徴に基づく方式であるShepherd Cacheにロード命令毎の特徴に基づく方式を組み合わせるキャッシュリプレース最適化手法を提案する。これにより、再参照されるまでの期間が長いエントリの特徴をロード命令毎に抽出し、このようなエントリの再参照間隔の予測値をリプレース対象の決定に利用できるようになる。

提案した手法の有用性を検証するため、従来のShepherd Cacheに提案モデルを実装し、SPEC CPU 2000を用いてシミュレーションにより評価した。その結果、既存のShepherd Cacheと比べてキャッシュミス率を最大19.4%、平均1.0%抑制することができた。

ロード命令およびエントリ毎の特徴に基づく 再参照間隔予測を用いたキャッシュ性能向上手法

目次

1	はじめに	1
2	背景	2
2.1	ロード命令毎の特徴を考慮したキャッシュ性能向上手法	2
2.1.1	Scan Bypassing	2
2.1.2	Dead Block 予測	3
2.2	エントリ毎の特徴を考慮したキャッシュ性能向上手法	3
2.2.1	Least Frequently Used	3
2.2.2	再参照間隔予測	3
2.3	OPT	4
2.4	Shepherd Cache	4
3	再参照間隔予測の精度向上によるキャッシュリプレイス最適化	8
3.1	従来手法の問題点	8
3.1.1	粗粒度な情報に基づく予測	8
3.1.2	キャッシュの利用効率の低下	9
3.1.3	SCでは再参照間隔を予測できないエントリ	11
3.2	ロード命令およびエントリ毎の特徴を考慮した再参照間隔予測	11
4	実装と動作モデル	13
4.1	追加ハードウェア	13
4.1.1	拡張したキャッシュの構成	13
4.1.2	PCTableの実装コスト	15
4.2	動作モデル	15
4.2.1	キャッシュヒット時の動作	15
4.2.2	キャッシュミス時の動作	18
4.2.3	PCTable参照に要するアクセスレイテンシ	20
5	評価	20
5.1	評価環境	21
5.2	Confidence Counter (CC) の設定	22

5.3	評価結果	23
5.4	考察	25
6	おわりに	29
	謝辞	30
	参考文献	30

1 はじめに

プロセッサの高速化は、主に集積回路の微細化による高クロック化によって実現されてきた。しかしながら、集積回路の微細化に伴い配線遅延がゲート遅延に対して相対的に増大し、配線遅延がプロセッサ性能に与える影響が大きくなってきた事により、高クロック化のみではプロセッサを高速化させる事が難しくなった。このため、配線遅延を隠蔽するための手法として、キャッシュメモリの高性能化が広く研究されてきた。キャッシュメモリを高性能化させるための手法は多岐に渡り、その動作をフルアソシアティブキャッシュに近づける手法 [1, 2], キャッシュエントリに優先度を設けてリプレースに利用する手法 [3, 4, 5, 6], データをプリフェッチする手法 [7, 8] など様々な手法が存在する。

これまでに提案されてきたキャッシュ性能向上手法は、ロード命令毎の特徴に基づく方式とエントリ毎の特徴に基づく方式の2つに大別できる。まず、ロード命令毎の特徴に基づく方式では、ロード命令毎にエントリの特徴を抽出し、この特徴を用いてリプレースの優先度を決定する。しかし、同じロード命令によってキャッシュ上に配置されるエントリでも、各エントリ毎に再参照される確率や間隔が大きく異なる場合があり、このような場合にはロード命令毎に特徴を抽出することは困難である。一方、エントリ毎の特徴に基づく方式では、キャッシュ上の各エントリの特徴を抽出し、この特徴を用いてリプレースの優先度を決定する。この方式では、ロード命令毎には抽出できなかった特徴を考慮することができるが、新たにキャッシュ上に配置されたエントリの特徴を抽出するためにキャッシュの一部をモニタリングに使用しなければならない。そのためキャッシュの利用効率が低下し、キャッシュの性能が低下する可能性がある。以上のように、これまでに提案されてきた手法にはそれぞれ利点と欠点が存在する。

様々なキャッシュ性能向上手法が提案されている一方で、最適なキャッシュリプレースアルゴリズムとして *Optimal Replacement (OPT)* [9] と呼ばれるアルゴリズムが存在する。しかし、これを実現するにはアクセスされるアドレスの順序情報を事前に知る必要があるため、実装は現実的ではない。そこで、最適なりプレースアルゴリズムである *OPT* を擬似的に実現する手法として **Shepherd Cache** (以下, **SC**) [10] が提案されている。本研究ではこの **SC** をベースとし、ロード命令およびエントリ毎の特徴に基づく再参照間隔予測によるキャッシュリプレース最適化手法を提案する。

以下、2章では従来のキャッシュリプレース最適化手法をロード命令毎の特徴に基づ

く方式とエン트리毎の特徴に基づく方式に分類して考察し、更に最適なりプレースアルゴリズムである OPT と、OPT を擬似的に実現している SC について述べる。3 章では、従来手法の問題点および提案手法の概要について述べ、4 章でその実装方法と動作モデルについて説明する。5 章では提案手法を評価・考察し、6 章で結論を述べる。

2 背景

本章では、より最適に近いキャッシュリプレース手法を考えるため、これまでに提案されてきたキャッシュ性能向上手法を 2 つの着眼点に基づき分類する。また、最適なキャッシュリプレースアルゴリズムである OPT と、OPT を擬似的に実装した SC について説明する。

2.1 ロード命令毎の特徴を考慮したキャッシュ性能向上手法

本研究では、ロード命令毎にエントリの特徴を調べてリプレースの優先度を決定する手法を、「ロード命令毎の特徴に基づく方式」として考察する。なお、この方式で決定した優先度を用いるキャッシュ性能向上手法としては、全く再参照されないエントリをキャッシュ上に配置しない Scan Bypassing や再参照される見込みのないエントリを優先的にキャッシュから追い出す Dead Block 予測などが挙げられる。以下では、この 2 つの手法について説明する。

2.1.1 Scan Bypassing

再参照されるまでの期間が非常に長いエントリへのアクセスが多発することで、多くのエントリが全く再参照されることなくキャッシュから追い出されてしまうことを **Scan**[11] と呼ぶ。Scan が発生した場合、再参照されることのないエントリばかりがキャッシュ上に配置されてしまうため、再参照可能なエントリが追い出されてしまう可能性がある。この問題を解決するために、Scan の発生をロード命令毎に検出し、再参照されないと予測されたデータを、キャッシュ上に配置せずにレジスタに直接書き込んだり、容量の小さい L1 キャッシュ上には配置せずに L1 キャッシュよりも大容量の L2, L3 キャッシュ上に配置する操作であるバイパッシングを行う手法が提案されている。堀部ら [12] は、ロードおよびストアの命令アドレス毎にキャッシュミス率を算出し、ミス率の高いエントリを L1 キャッシュ上には配置せずに L2 キャッシュ上に配置することでキャッシュの性能を向上させる手法を提案している。

2.1.2 Dead Block 予測

この手法では、再参照されないと予測されたエントリを **Dead** とし、キャッシュから優先的に追い出すことでキャッシュの性能を向上させる。なお、Dead となるエントリの検出にロード命令毎の特徴を用いる方式は多く存在するが、Lai ら [13] が提案する予測器では、過去のアクセスパターンから Dead を早期にキャッシュから追い出す。また、Khan ら [14] は、一度も再参照されず Dead になると予測したエントリをバイパスングする事で更なる性能向上を図っている。

2.2 エントリ毎の特徴を考慮したキャッシュ性能向上手法

本研究では、キャッシュ上の各エントリの特徴を調べてリプレースの優先度を決定する手法を、「エントリ毎の特徴に基づく方式」として考察する。なお、この方式で決定した優先度を用いるキャッシュ性能向上手法としては、Least Frequently Used (LFU) や再参照間隔予測が挙げられる。以下では、この2つの手法について説明する。

2.2.1 Least Frequently Used

Lee ら [3] が提案する LFU 方式では、エントリ毎に参照頻度を記録し、この参照頻度が最も低いエントリを優先的にリプレースする。これにより、一度しか参照されないアドレスへのアクセスが多発するストリーミング処理や、キャッシュ上に配置されたエントリが再参照される前に追い出されてしまうスラッシングの発生によるキャッシュの性能低下を回避する事ができる。また、この手法では、LFU 方式と Least Recent Used (LRU) 方式を組み合わせることで、今後参照頻度が高くなる可能性のある、直近に参照されたエントリがリプレースされてしまうことを回避している。

2.2.2 再参照間隔予測

Aamer ら [5] が提案する Re-Reference Interval Prediction (RRIP) では、エントリ毎の特徴を用いてキャッシュ上のエントリが再参照される間隔を予測する。図1に示すように、新たなエントリをリプレースの優先度がある程度高い位置へと挿入し、キャッシュヒットしたエントリは LRU 方式と同様に最もリプレースの優先度が低い位置へと移動させる。このようにすることで、図1における灰色の領域には、キャッシュ上に配置されてから一度でも再参照された事がある**既参照エントリ**のみが存在するようになり、既参照エントリはキャッシュに配置されてから一度も再参照された事がない**未参照エントリ**よりも優先される。このため、ストリーミング処理やスラッシングの発生による性能低下を抑制できる。

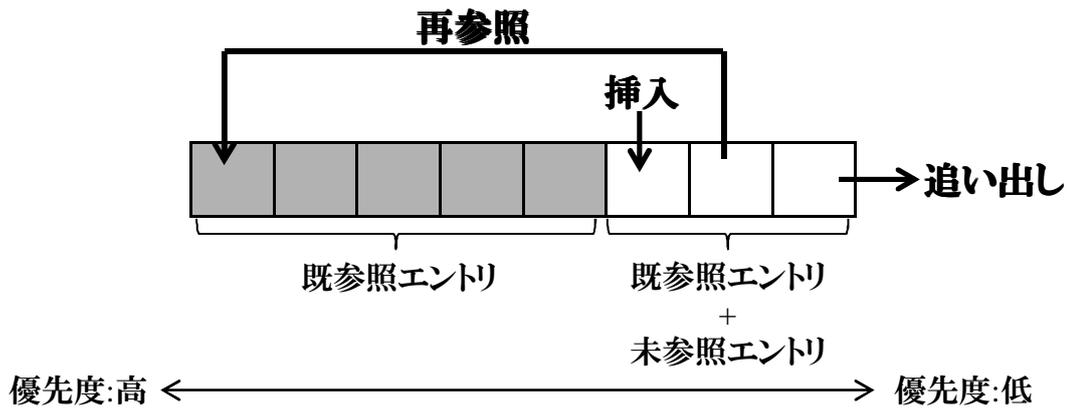


図 1: RRIP の動作

2.3 OPT

最適なりプレースアルゴリズムとして、再参照されるまでの期間が最も長いエントリを追い出す方式である OPT[9] が知られている。しかしながら、OPT に基づくキャッシュを実現するためにはアクセスされるアドレスの順序情報を事前に知る必要があるため、実装は現実的には困難である。このため、一般的なキャッシュメモリで利用されているリプレースアルゴリズムは、再参照されるまでの期間が最も長いエントリを予測する事でエントリのリプレース順を決定している。例えば、LRU 方式では、最近使われていないエントリが最も未来に参照されると予測する。しかしながら、このような予測を用いた場合、ストリーミング処理を含むプログラムが実行された際に有用なエントリが再参照されないエントリによって追い出されてしまう。また、非常に大きなワーキングセットを持つプログラムが実行された場合、スラッシングが発生する。このように、LRU 方式では、特定のアクセスパターンに対して、その性能が大きく低下してしまう。

2.4 Shepherd Cache

前節で述べたように最適なりプレースアルゴリズムである OPT の実現は困難なことから、擬似的に OPT を実現する手法として、Rajan ら [10] は Shepherd Cache を提案している。この手法では、アクセスされるアドレスの順序情報の代わりに再参照間隔予測を用いてリプレース対象を決定する。本節では、SC の具体的な実装と動作モデルについて説明する。

SC では、擬似的に OPT を実現するために LRU 方式のセットアソシアティブキャッ

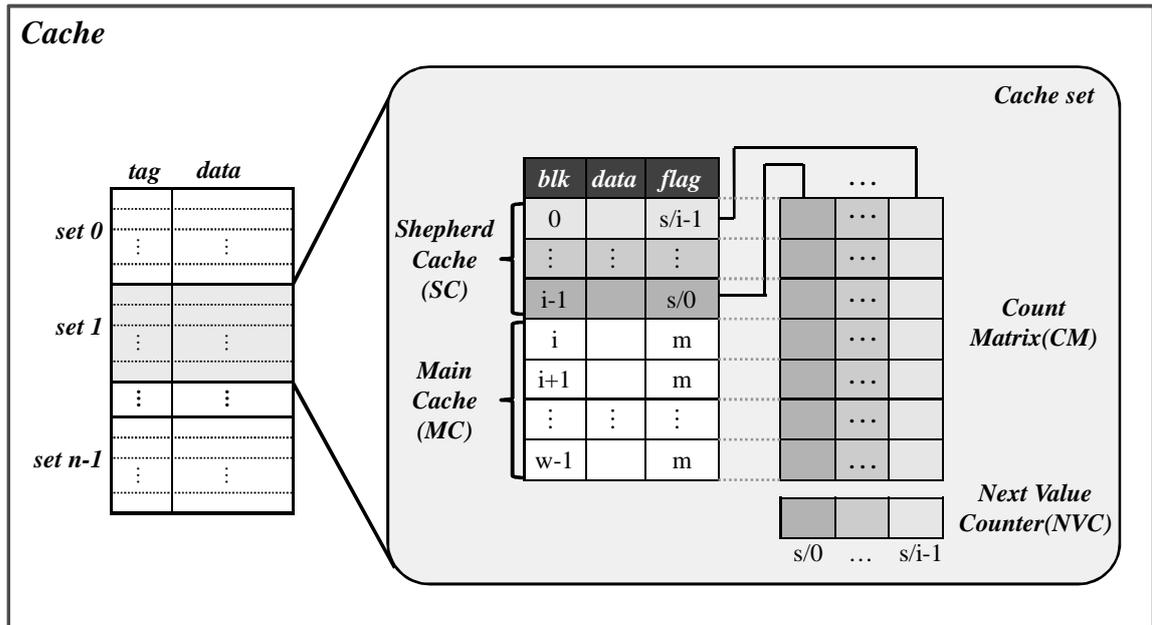


図 2: Shepherd Cache のハードウェア構成

シユにエントリの再参照間隔を予測する機構を追加する. SC を実装したキャッシュの構成を図 2 に示す. なお, ここでは w ウェイセットアソシアティブキャッシュとし, セット数は n であるとする.

まず, キャッシュの各セットをモニタリングに使用される **Shepherd Cache (SC)** と再参照間隔予測でリプレースされる **Main Cache (MC)** の 2 つの領域に分割する. SC には新たにキャッシュに挿入されたエントリが配置され, First-In-First-Out (FIFO) でリプレースされる. ここでは, 各キャッシュセットのうち i ウェイを SC としている. また, 各キャッシュブロックが SC か MC であるかを区別する **flag** を追加する. なお, このフラグのビット幅は $\log_2(i+1)$ bits となる.

更に, エントリ毎の特徴を抽出するため, あるキャッシュミスが発生してから各キャッシュブロックがどのような順番でキャッシュヒットしたかを記録するテーブルである **Count Matrix (CM)** と, キャッシュヒットしたエントリ数をカウントする **Next Value Counter (NVC)** を追加する. 図 2 における CM の横のラインはキャッシュ上の各ブロックに, 縦のラインは SC の各ブロックに関連付けられている. また, NVC も SC の各ブロックに関連付けられている. この手法では, キャッシュヒットした際, ヒットしたブロックの CM に NVC の値を挿入することでヒット順を記録し, リプレース対象を選択する際には CM の値を再参照間隔の予測値として用いる. また, NVC

は CM に値が挿入されてからカウントアップされる。なお、CM 全体のビット幅は $\log_2\{w \times i \times (i + 1)\}$ bits, 各 NVC のビット幅は $\log_2 i$ bits となる。

図 3 を例に、実際の SC の動作を説明する。なお、例では n 番目のブロックを $\text{blk}n$ と表し、図の下部に示す Access sequence の順にキャッシュにアクセスする状況を仮定している。まず、図 3(a) に示すように SC にエントリが入っていない状態を初期状態とする。この状態で G に対するアクセスが発生すると、G はキャッシュ上に存在しないことから、キャッシュミスとなる。この時、まだエントリが配置されていない SC に新たなエントリが挿入され、このブロックに対する NVC が 0 に、CM は「まだ予測値が入っていないことを表す値」に初期化される (図 3(b))。次に、A に対するアクセスが発生すると、 $\text{blk}2$ に A が存在することから、キャッシュヒットとなる。この時、SC である $\text{blk}1$ にエントリが挿入されてから $\text{blk}2$ がどのような順番でヒットしたかを記録するために、 $\text{blk}2$ の CM に $\text{blk}1$ に対する NVC の値が挿入される。その後、次にヒットしたエントリのヒット順を識別するために、NVC はカウントアップされる (図 3(c))。なお、SC 上のエントリに対するアクセスが発生した場合にも、同様の操作を行う (図 3(d))。

次に、H に対するアクセスが発生した場合、キャッシュミスとなり、G の時と同様に $\text{blk}0$ に対する NVC と CM の値が初期化される。このとき、モニタリングに使用している SC 上のエントリは、モニタリングが終わるまでキャッシュ上に残しておく必要があるため、リプレースの優先度を下げる。ここで、この手法では CM の値が最大となるエントリを再参照間隔が最も広いエントリとみなしてリプレースする。この手法では、既にエントリが配置されている SC である $\text{blk}1$ において、 $\text{blk}0$ に対する CM の値を 0 にし、 $\text{blk}0$ が挿入されてからすぐにヒットしたエントリとみなす。このことで、 $\text{blk}1$ は $\text{blk}0$ に対する再参照間隔が短いエントリとして扱われるため、リプレースの優先度が下がり、キャッシュからリプレースされにくくなる (図 3(e))。次に、C に対するアクセスが発生した場合は A の時と同様に CM と NVC を更新し、以降もキャッシュヒットが続くため、同様の操作を行う (図 3(f)(g))。

最後に、I に対するアクセスが発生すると、キャッシュミスとなり、キャッシュ上のいずれかのエントリがリプレースされる。この場合、新たなエントリが挿入された後のキャッシュヒット順をモニタリングするために、新たなエントリはモニタリング領域である SC に挿入する。しかし、全ての SC には既にエントリが配置されているため、一番最初に SC 上に配置されたエントリを FIFO に基づいて SC から追い出し、新たなエントリを挿入する。なお、SC から追い出されたエントリは、フラグを書き換えるこ

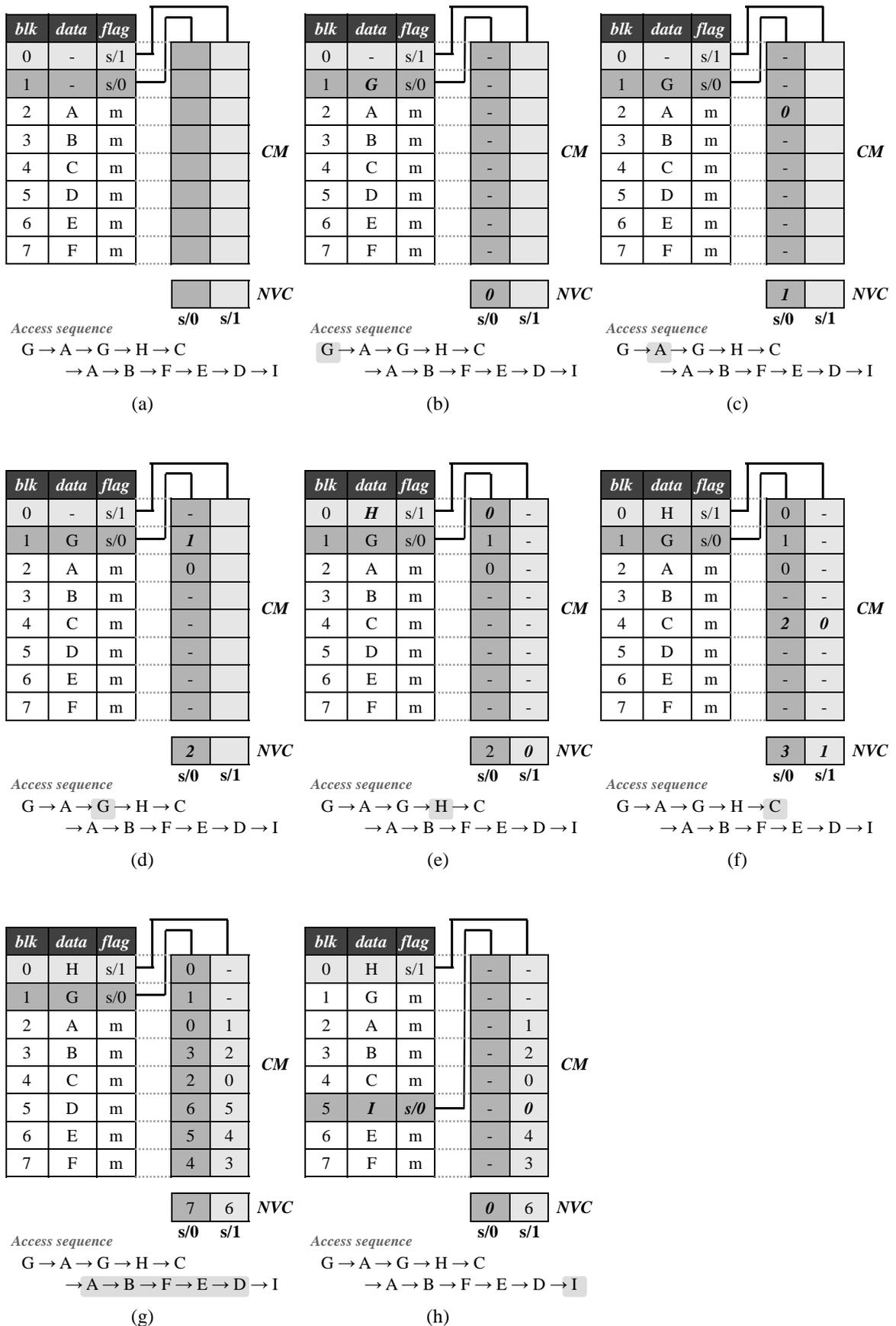


図3: SCの動作モデル

とで以降 MC として扱う。また、この時に SC から追い出されるエントリの CM の値を用いて再参照間隔を予測し、リプレース対象を決定する。この例では、SC のブロックのうち blk1 が最初に挿入されていることから、blk1 に対する CM の値が最大となるエントリを再参照間隔が最も広いエントリであるとみなしてリプレースする。例では図 3(h) に示すように、blk5 の D がリプレース対象として選択される。また、新たにエントリが挿入された blk5 の flag を s/0 に、SC から追い出された blk1 の flag を m に書き換え、SC として使用されていたエントリと MC として使用されていたエントリの用途を入れ替える。更に、新たに挿入されたエントリに対する NVC と CM を初期化する。

以上のような動作で、SC では擬似的に OPT を実現する。ここで、この手法を実現するために必要なアクセスレイテンシについて説明する。キャッシュミスが発生した場合、この手法ではリプレース対象となるエントリを決定し、SC と MC のエントリを入れ替え、CM や NVC を初期化する必要がある。しかし、これらの処理はメインメモリへのアクセスが終わるまでに終了していればよい。ここで、メインメモリへのアクセスは、一般的に 100 サイクルから 200 サイクル程度のレイテンシを要する。これはキャッシュへのアクセスに必要なサイクル数と比べ非常に大きく、CM や NVC の更新に時間がかかったとしても、データを要求されてから当該データを返すまでのルックアップ速度には影響しないと考えられる。そのため、この手法は LRU 方式のセットアソシアティブキャッシュと同程度のルックアップ速度を保ったまま実現することができる。なお、この手法はエントリ毎に特徴を調べて再参照間隔を予測していることから、エントリ毎の特徴に基づく方式に分類することができる。

3 再参照間隔予測の精度向上によるキャッシュリプレース最適化

本章では、以上で述べた従来手法の問題点を挙げる。また、これらの問題点を解決するキャッシュリプレース手法を提案する。

3.1 従来手法の問題点

2.1 節、2.2 節、および 2.4 節で説明した従来手法には、それぞれ利点と欠点がある。本節では、これらの手法の問題点について述べる。

3.1.1 粗粒度な情報に基づく予測

ロード命令毎の特徴に基づく方式では、ロード命令毎にエントリの特徴を抽出することで再参照間隔を予測できる。しかし、同じロード命令によってキャッシュ上に配

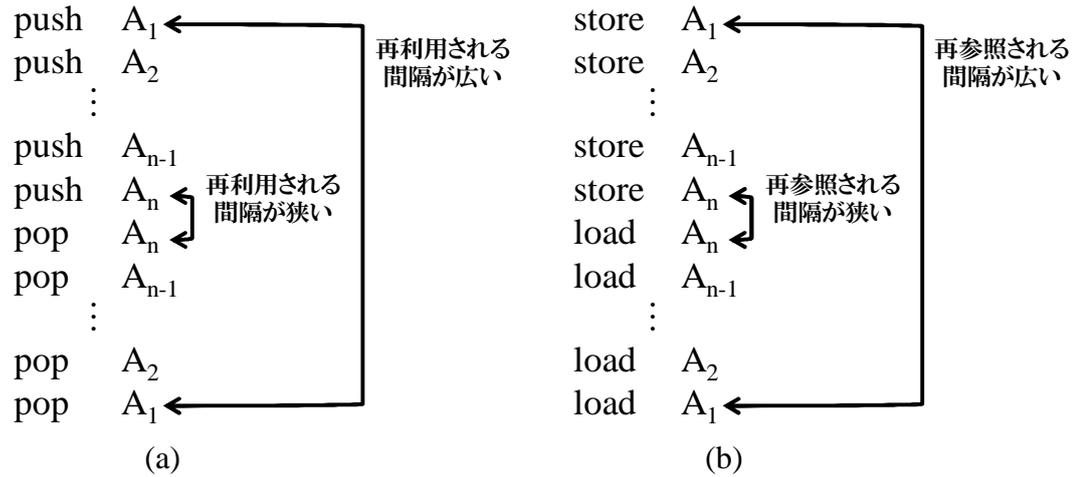


図 4: 典型的なスタックアクセスパターン

置されるエントリでも、エントリ毎に再参照される確率や間隔が大きく異なる場合がある。例えば、典型的なスタックアクセスパターンでは、図 4(a) に示すように任意の n 個のデータが push された後 pop される。スタックに対する push/pop 操作は一般に、あるメモリ領域に対する store/load 命令で実現されるため (図 4(b))、それぞれにおいて対象のデータにメモリアクセスが発生するが、図 4 に示すような操作をする際、キャッシュのウェイ数が n 未満の場合、スタックの底辺近くに格納されている A_1 や A_2 へのアクセスが発生してから A_n がアクセスされるまでの間に、 A_1 や A_2 のデータを持つエントリはキャッシュから追い出されてしまう。また、キャッシュのウェイ数が n 以上の場合でも、スタックでは First-In-Last-Out でデータが利用されるため、スタックの浅い位置に格納されたデータほど再び利用される間隔は狭くなり、底辺近くに格納されたデータほど再び利用されるまでの間隔が広くなる。このため、スタックに挿入されている位置によってそのデータを持つエントリへのアクセスの間隔は異なる。

このような特徴は、キャッシュエントリを関連するロード命令毎にまとめて扱った場合には抽出することはできない。このように、ロード命令毎の特徴に基づく方式では、比較的粗粒度な予測しかできず、上述したスタックアクセスの例のように、十分な精度が達成できない場合がある。

3.1.2 キャッシュの利用効率の低下

エントリ毎の特徴に基づく方式では、キャッシュ上の各エントリの特徴を抽出するため、ロード命令毎の特徴に基づく方式よりも細粒度な情報に基づく予測が可能である。

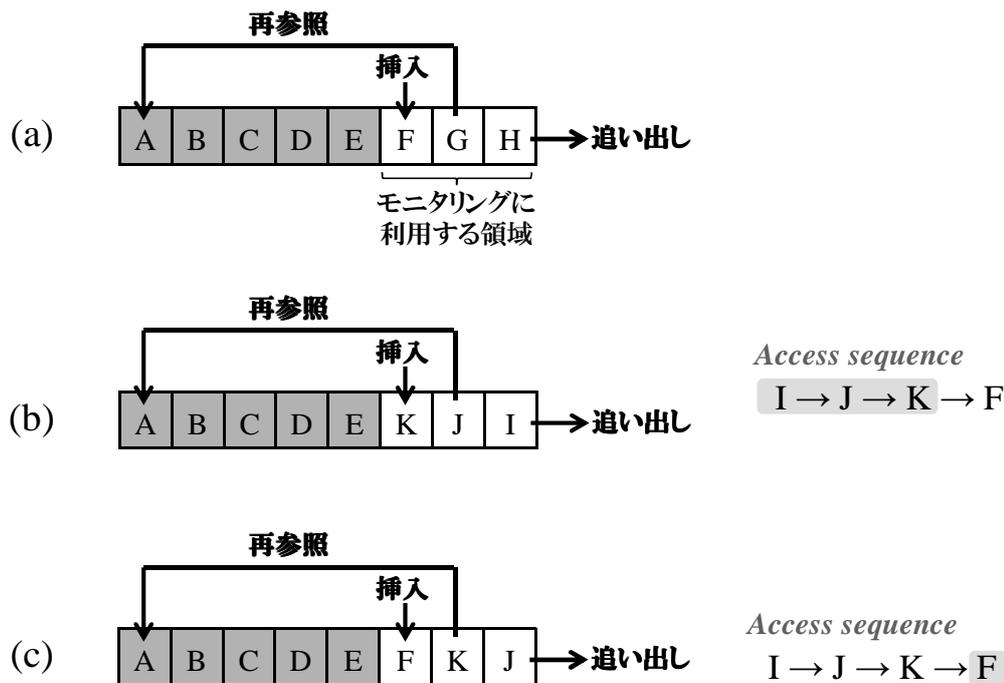


図5: モニタリング領域によってキャッシュの性能が低下する場合

しかし、キャッシュ上に新たに配置されたエントリの特徴を判断するためには、キャッシュの一部をモニタリングに利用し、そのエントリの特徴を抽出しなくてはならない。これについて、RRIPを適用したキャッシュにおいて図5に示すようなアクセスが発生した場合を例に説明する。RRIPでは、図5(a)に示すように再参照されたエントリをリプレースの優先度が低い位置に移動させ、白色の領域で新たにキャッシュに配置されたエントリのモニタリングを行う。まず、図5(a)のキャッシュ状態においてI, J, Kへのアクセスが発生すると、キャッシュ上のエントリはH, G, Fの順でリプレースされる(図5(b))。次に、図5(c)に示すようなFへのアクセスが発生すると、FはKへのアクセスが発生した時にリプレースされているため、リプレースされた直後に再び挿入されることになる。このように、エントリ毎の特徴に基づく方式ではモニタリングに利用する領域に新しいエントリを配置するため、モニタリング領域の広さ次第では、再参照されるエントリでも、再参照されると判断される前にリプレースされてしまう可能性がある。また、再参照されることのないエントリばかりがキャッシュ上に配置されてしまうストリーミング処理やScanが発生し、このようなアクセスパターンの中に再参照されるエントリが混ざっているような場合、再参照されないデータをバイパスすることで、再参照されるエントリをキャッシュヒットさせることができ

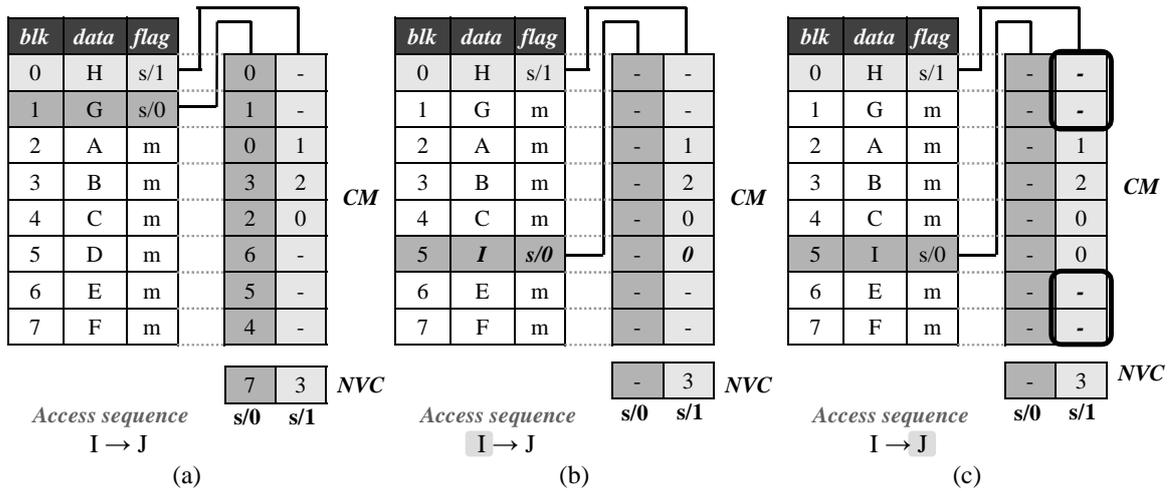


図6: SCではリプレース対象を決定できないキャッシュ状態

る。しかし、エントリ毎の特徴に基づく方式では、キャッシュ上にエントリを配置する前にその特徴を抽出することはできず、このようなアクセスパターンに対して性能を向上させることは難しい。

3.1.3 SCでは再参照間隔を予測できないエントリ

エントリ毎の再参照間隔予測を用いて擬似的にOPTを実現しているSCでは、SCに新たなエントリが挿入されてから、どのような順番でキャッシュ上のエントリにアクセスがあったかをCMに記録している。しかし、キャッシュ上のエントリの中には、SCにエントリが挿入されてから一度もキャッシュヒットしないものも存在し、このようなエントリのCMには予測値が挿入されない。このため、リプレース対象を選択する際、全てのエントリに対して再参照間隔を予測できない可能性がある。これについて、図6に示すようなキャッシュアクセスが発生した場合を例に説明する。図6(a)で示す状況において、新たにIをキャッシュに挿入する場合は、CMの値からリプレース対象を決定することができる(図6(b))。しかし、続いてJをキャッシュに挿入しようとすると(図6(c))、CMに予測値が格納されていないエントリが複数存在するため、このようなエントリの中からLRU方式に基づいてリプレース対象を決定しなければならない。そのため、図6の例のようなキャッシュミスが多く発生するようなアクセスパターンに対し、SCによるキャッシュの性能向上は見込めない。

3.2 ロード命令およびエントリ毎の特徴を考慮した再参照間隔予測

2章で説明したように、これまでロード命令毎の特徴またはエントリ毎の特徴を考慮した手法が提案されてきた。しかし、これら両方の特徴を考慮した手法については深く考察されていない。また、従来手法にはそれぞれ利点と欠点がある。そこで、ロード命令毎の特徴とエントリ毎の特徴の両方を考慮することで、3.1節で挙げた従来手法の問題点を改善できると考えられる。本研究では、エントリ毎の特徴を用いて再参照間隔を予測するSCに対して、ロード命令毎の特徴に基づく再参照間隔予測手法を組み合わせた手法を提案する。

3.1.3節で述べた通り、従来手法ではあるキャッシュミスが発生してからキャッシュヒットしていないエントリのCMの値は更新されない。つまり、SCでは短期間によくヒットするエントリの再参照間隔を予測することはできても、ヒットまでの期間が長く、長期間のサンプリングが必要なエントリの再参照間隔を予測することは困難である。また、SCに新たなエントリが挿入されると、このエントリが挿入されてからのキャッシュヒット順を記録するために、このエントリをキャッシュに挿入する前に記録されていたヒット順の情報はリセットされてしまう。しかし、過去のキャッシュヒットと同じような傾向でヒットするエントリがあった場合、過去に収集したCMの値がリプレース対象の選択に有用な再参照間隔の予測値である可能性がある。そのため、過去に記録したCMの値を使用することができれば、長期間のサンプリングが必要なエントリの再参照間隔を早期に予測できると考えられる。

そこで、過去のキャッシュヒットの傾向としてロード命令毎にCMの値を記録しておき、これをロード命令毎の特徴として利用する再参照間隔予測手法を提案する。これについて、図7を用いて説明する。この図では、従来のSCにロード命令毎にCMの値を記録するテーブルを追加しており、図6と同様のキャッシュアクセスが発生すると仮定している。まず、図7(a)の状態において、CMの値をロード命令毎に記録しておく。その後、Iへのアクセスはキャッシュミスとなり、キャッシュ上のいずれかのエントリをリプレースする。このとき、従来のSCと同様にCMの値からリプレース対象を決定する(図7(b))。これに続いてJへのアクセスが発生すると、CMに値が格納されていないために再参照間隔を予測できないエントリがあるため、この例ではテーブルに記録されている過去に収集したCMの値をblk1に対するCMに挿入する(図7(c))。このように、過去に収集した特徴を利用することにより、図6で再参照間隔が予測できなかったエントリについても再参照間隔を予測できるようになる。このとき、これらのエントリが過去と同じ傾向でキャッシュヒットするものであれば、LRU方式よりも最適に近いキャッシュリプレースが実現できる。

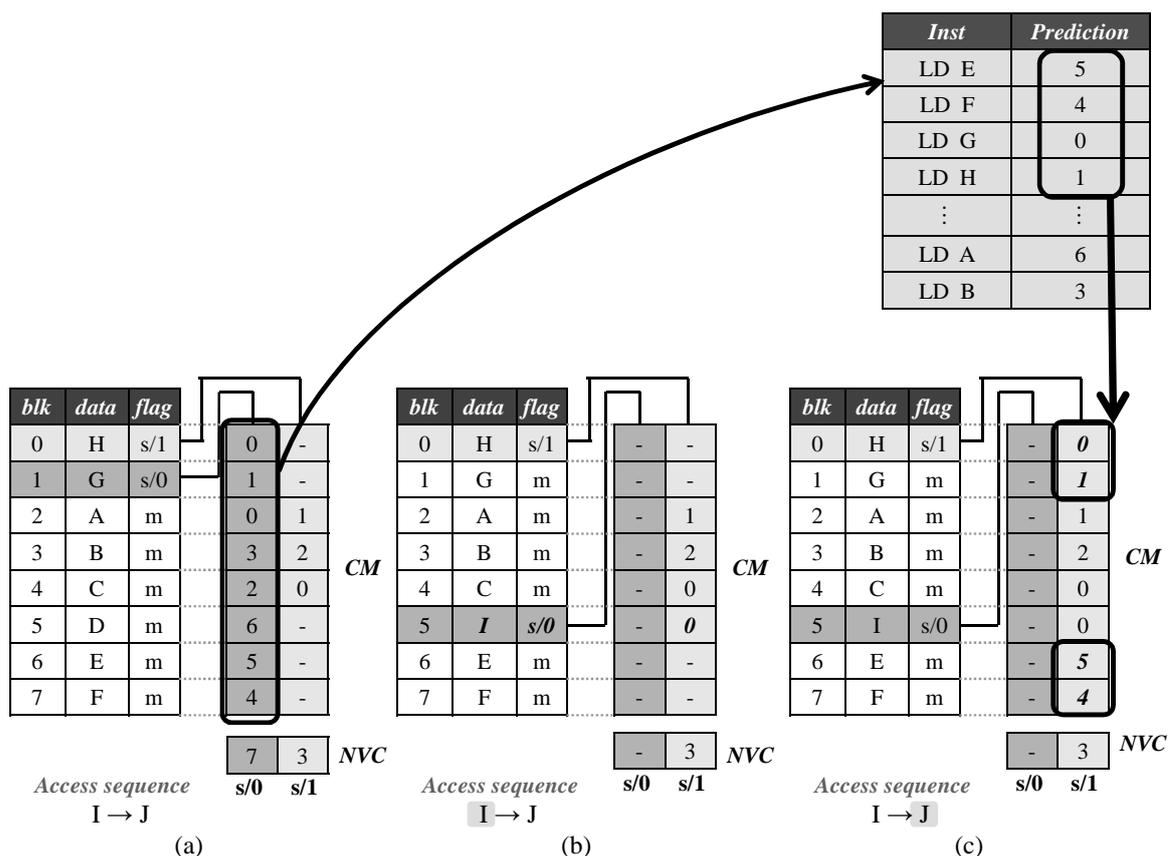


図7: 過去に収集した特徴を利用した場合の動作

4 実装と動作モデル

本章では、提案手法を実現するための具体的な実装方法を述べる。更に、その動作モデルについても述べる。

4.1 追加ハードウェア

まず、提案手法の実現に必要なハードウェアの構成について説明する。また、追加したハードウェアの実装コストについて考察する。

4.1.1 拡張したキャッシュの構成

ロード命令毎の特徴に基づく再参照間隔予測を行うために、キャッシュに **PCTable** と呼ぶ表を追加する。PCTable の各ラインは、以下の4つのフィールドを持つ。

index (idx) :

キャッシュから PCTable を参照するために用いるインデックス番号。

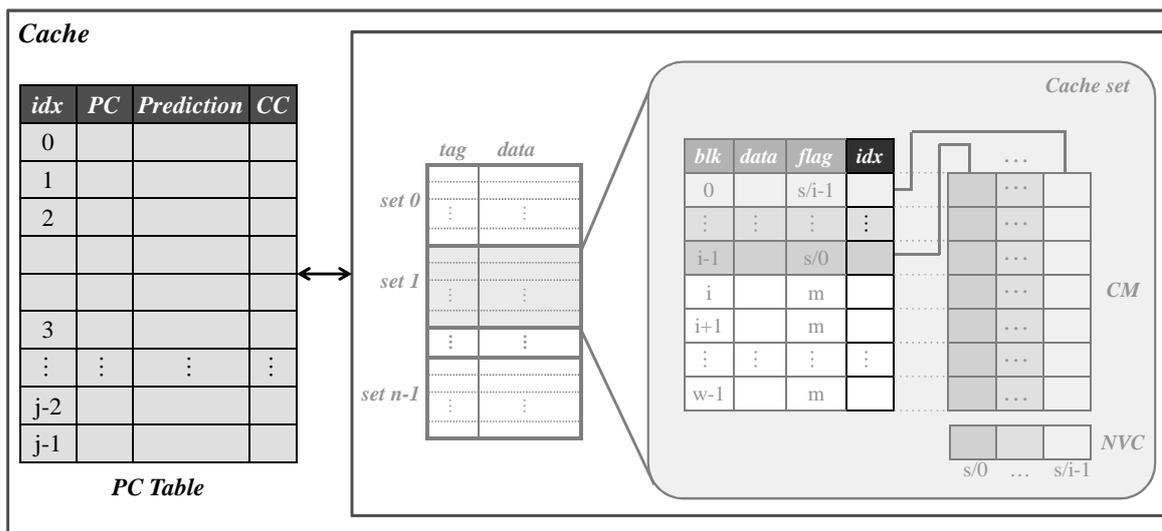


図 8: 拡張した L2 キャッシュの構成

Program Counter (PC) :

過去に実行されたロード命令のプログラムカウンタを記録する。ビット幅は 64 bits とする。

Prediction :

ロード命令毎の特徴に基づく再参照間隔の予測値を記録する。

Confidence Counter (CC) :

同じロード命令によってキャッシュ上に配置されるエントリが過去と同じ傾向で再参照されるかを調べる飽和カウンタ。

以上のフィールドを持つ PC Table を追加したキャッシュの構成図を、図 8 に示す。なお、キャッシュ構成は w ウエイのセットアソシアティブとし、セット数は n であるとする。また、SC は i ウエイとし、PC Table のエントリ数は j であるとする。

本提案手法では、過去に実行されたロード命令のプログラムカウンタを PC に登録し、そのロード命令によりキャッシュ上に配置されたエントリの CM を、ロード命令毎の再参照間隔の予測値として Prediction に記録する。また、その予測値が信頼できるかを判断するために、Confidence Counter (CC) を用意する。この手法では、キャッシュヒットした際に、ヒットしたエントリの CM の値と、そのエントリに対応する PC Table のラインの Prediction とを比較し、一致した場合には、そのロード命令によってキャッシュ上に配置されるエントリは過去と同じ傾向で再参照されるとみなす。このとき、

CCをカウントアップする。また、SCだけではリプレース対象を決定できない場合に、CCが閾値以上となっているラインのPredictionの値のみを再参照間隔の予測値として用いる。

なお、PCTableのPredictionに値を挿入したり、キャッシュミス時にリプレース対象を決定する際に、キャッシュからPCTableを参照するが、この参照にはPCTableの各ラインに付けられているインデックス番号を用いる。このため、各キャッシュブロックにも、そのブロックに対応するPCTableのラインのインデックス番号を記録しておくフィールドを追加し、このインデックス番号は、そのブロックに新たにエントリが挿入された時に更新する。

4.1.2 PCTableの実装コスト

以上のような機構をキャッシュに追加する場合の実装コストについて考察する。まず、PCTableのインデックス番号を表現するために必要なビット幅は、最大でも $\log_2 j$ bitsである。また、PredictionにはCMの値を挿入するため、その値を記憶するために必要なビット幅は $\log_2 i$ bitsである。これに加え、CCは飽和カウンタを利用して実装するため、その実現に必要なビット幅は $\log_2 \{\text{CCの閾値}\}$ bitsとなる。なお、キャッシュミスを頻発する命令は、単一のプログラムでは10個程度かそれよりも少ない数である事が知られており[7]、PCTableの実装コストを削減することは比較的容易であると考えられる。例えば16ウェイのセットアソシアティブキャッシュにおいて、各セットのSCのウェイ数を8とし、PCTableのライン数を512、CCの閾値を4とすると、PCTableのラインあたりのビット幅は80 bitsとなる。この場合、テーブル全体でも実装コストは5KB程度であり、これは一般的に用いられるL1キャッシュの容量である32KBや64KBよりも十分に小さい。なお、本提案手法では理想的な性能を知るため登録可能なロード命令数の制限を無くした状態で評価する。

4.2 動作モデル

本節では、図9に示す順番でロード命令が実行された場合を例に、提案手法を実装したSCの動作を説明する。なお、CCの閾値は4とする。また、動作モデルを示した後、PCTableの参照に要するアクセスレイテンシについて考察する。

4.2.1 キャッシュヒット時の動作

まず、図10(a)で示す状況において、図9に示すロード命令でEに対するアクセスが発生したとする。この場合、Eはキャッシュ上に存在することから、従来のSCと同様にblk6に対するCMを更新し、CMに値を挿入したNVCをカウントアップする（図

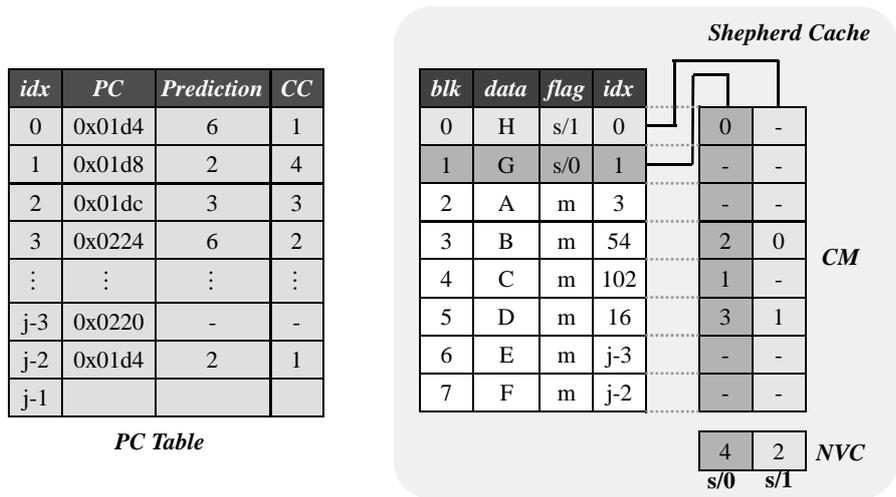
<i>PC</i>	<i>Inst</i>
0x0220	load E
0x0224	load A
0x02dc	load I
⋮	⋮



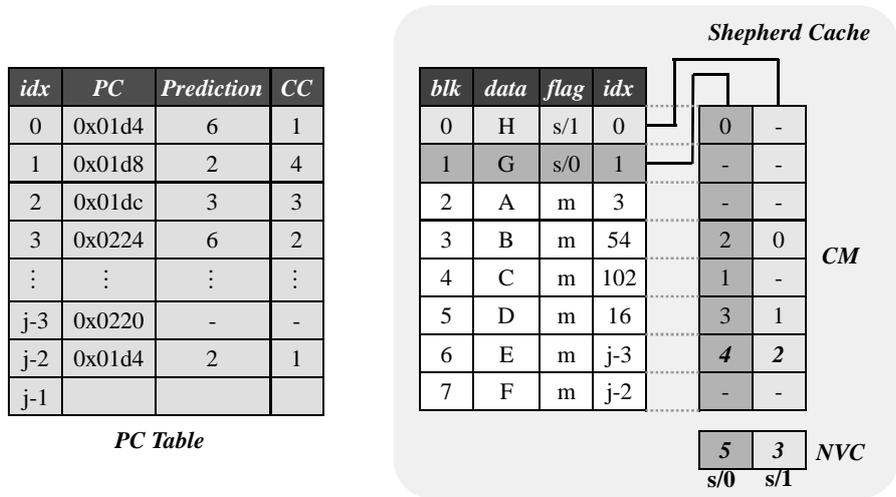
図 9: 実行されるロード命令

10(b)) . その後、インデックス番号を用いて PCTable を参照して、E をキャッシュ上に配置したロード命令の特徴を参照する (図 10(c)) . この時、このロード命令に対応する PCTable のラインには Prediction 値が挿入されていないため、blk6 の CM のうち最大のものを PCTable の Prediction 値として登録し、CC の値を 0 に初期化する.

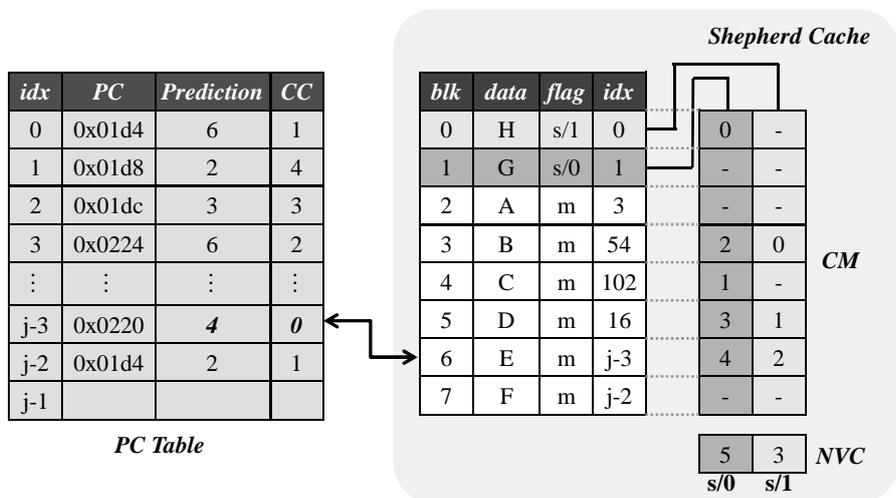
続いて、A に対するアクセスが発生したとする (図 11) . この時、A はキャッシュ上に存在するため、blk2 に対する CM と NVC を更新した後、PCTable を参照する. この場合、A に対応する PCTable のラインの Prediction には値が挿入されているので、このプログラムカウンタが指すロード命令によってキャッシュ上に配置されるエントリーは過去と同じ傾向で再参照されるあるかを調べるために、PCTable の Prediction 値と blk2 の CM の値を比較する. ここで、CM の値は前回リプレースが発生してから各ブロックが参照された順番であり、その順番が入れ替わることは多いと考えられる. このため、これらの値が一致した場合のみにその予測値が信頼できるものと判断すると、厳密に同じ間隔で参照されるエントリーの CM の値しかロード命令毎の予測値として使用することができず、提案手法による性能の向上は見込めない. そこで、本提案手法では図 12 に示すように、該当のロード命令によってキャッシュ上に配置されたエントリーが「最悪でもこの順番までにはキャッシュヒットする」ことを予測し、収集する情報の粒度を粗くする. つまり、Prediction 値と、該当エントリーの CM のうち最大の値とを比較し、Prediction 値より CM の最大値が小さい場合にロード命令毎の特徴に基づく再参照間隔の予測値は正しいとみなして CC をインクリメントする. Prediction 値よりも CM の値が大きい場合、CC が 0 であれば参照元のエントリーの CM のうち最大の値を Prediction 値とし、CC が 0 以上であれば CC をカウントダウンする. 図 11 の例では、blk2 の CM の最大値である 5 が Prediction の値よりも小さいので CC をカウントアップしている.



(a)



(b)



(c)

図 10: Prediction の更新

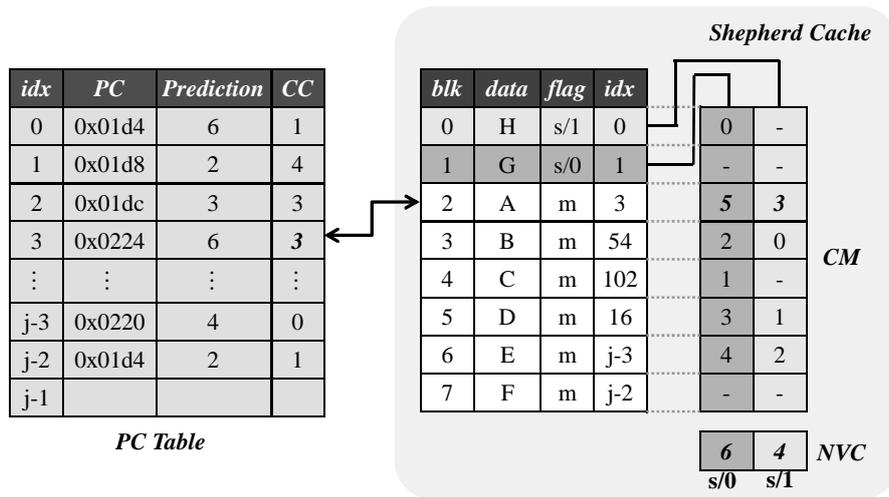


図 11: CC の更新

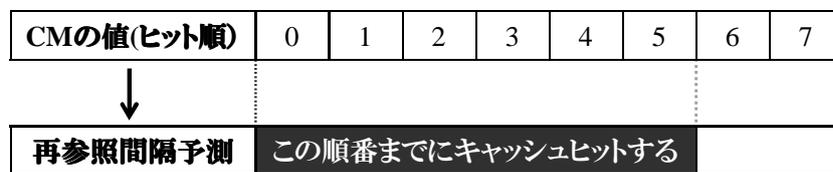
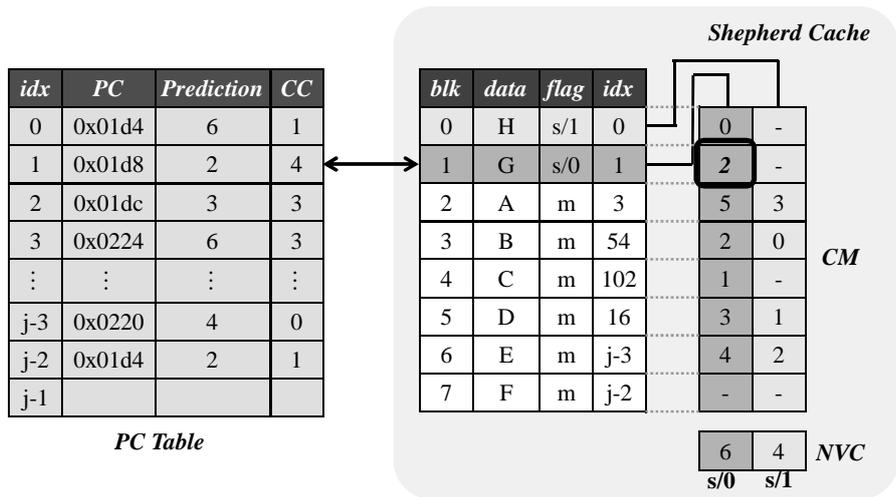


図 12: 提案手法におけるロード命令毎の特徴に基づく再参照間隔予測

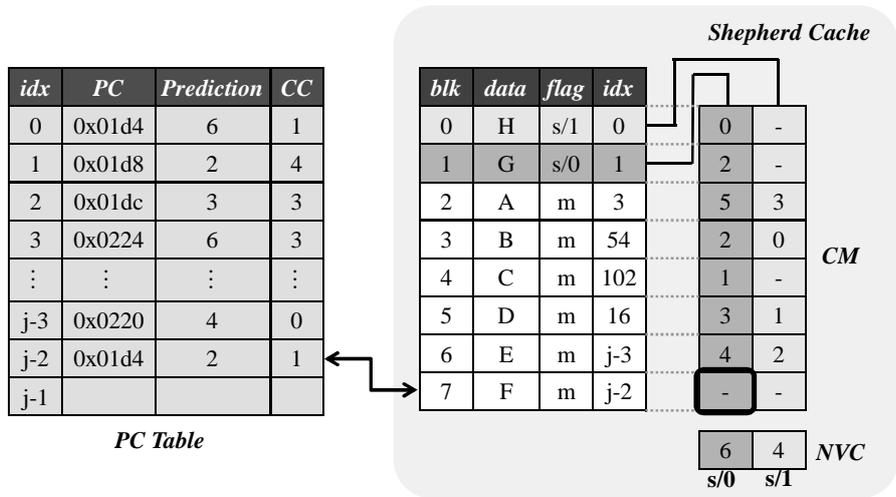
4.2.2 キャッシュミス時の動作

次に、キャッシュミス時の動作を説明する。図9に示したようにE, Aへのアクセスが発生した後、Iへのアクセスが発生したとする。この時、blk1に対するCMに値が挿入されていないエントリが存在するため、そのようなエントリに対応するPCTableを参照し、ロード命令毎の特徴に基づく再参照間隔の予測値を調べる(図13(a))。このラインのCCは閾値である4と等しいため、GはPredictionに格納されている再参照間隔までに再参照されるエントリであるとみなされ、CMにPredictionの値を挿入する。次に、blk7を見ると、このブロックもblk1に対するCMの値が入っていないため、同様の動作でblk7に対応するPCTableのラインを参照する(図13(b))。この時、blk7に対応するラインのCCの値は閾値を下回っていることから、このプログラムカウンタが指すロード命令によってキャッシュ上に配置されたエントリは、過去と同じ傾向で再参照されるとはみなされず、このPrediction値は再参照間隔の予測値として使用しない。

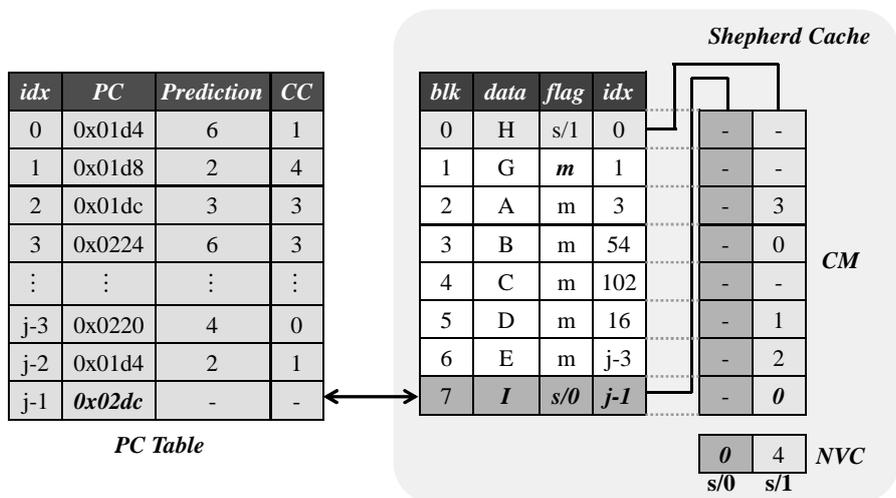
以上のようにPCTableからロード命令毎の特徴に基づく再参照間隔の予測値を調べ、



(a)



(b)



(c)

図 13: キャッシュミス時のリプレース対象選択

リプレース対象を決定する。その後、全てのエントリに対して再参照間隔の予測値が存在する場合、最も再参照間隔の広いエントリをリプレース対象とする。この一方で、ロード命令毎の特徴を調べた後でも CM に値が格納されていないエントリがある場合、従来手法と同様に LRU 方式を用い、CM に値が格納されていないエントリの中からリプレース対象を決定する。この例では、図 13(c) に示すように blk7 のエントリがリプレース対象となる。この時、従来手法と同様に SC と MC の用途を入れ替え、CM と NVC を初期化する。また、PCTable から新たなエントリをキャッシュに配置したロード命令のプログラムカウンタを探索する。この時、該当のプログラムカウンタが既に PCTable に登録されているならば、新たにエントリが挿入されたブロックには、そのエントリをキャッシュ上に配置したロード命令のプログラムカウンタが登録されているラインのインデックス番号を記録しておく。この一方で、該当するプログラムカウンタが見つからなければ、そのプログラムカウンタを PCTable に登録する。

4.2.3 PCTable 参照に要するアクセスレイテンシ

本提案手法では、従来手法で発生する CM へのアクセスに加えて PCTable へのアクセスが必要となる。特にキャッシュミス時には、実行されたロード命令のプログラムカウンタが PCTable に登録されているかを探索する必要がある。しかし、PCTable の探索操作はメインメモリへアクセスしている間に終了していればよい。2.4 節でも述べたように、メインメモリへのアクセスには 100 サイクルから 200 サイクル程度のアクセスレイテンシを要するため、この間に PCTable の探索操作は終了できると考えられる。このため、提案手法は LRU 方式のセットアソシアティブキャッシュと同程度のルックアップ速度を保ったまま実現することができる。しかし、PCTable が 512 ラインのフルアソシアティブ構成の場合、キャッシュミス時に全てのラインを探索するコストは大きい。このコストを削減するために、ハッシュ関数を用いるなどして PCTable の構成を変える必要がある。なお、本提案手法では、理想的な性能を知るために全てのラインを探索すると仮定して評価を行う。

5 評価

提案手法の有効性を確認するため、シミュレーションにより評価を行った。本章では、提案手法の性能について考察する。

表 1: シミュレータ諸元

Processor	
ISA	Alpha
number of cores	1 core
issue order	out-of-order
issue width	8
branch pred	8 KB g-share
LSQ	32 entry
ROB	64 entry
L1 I&D cache	
	16 KB
ways	2 ways
latency	2 cycles
Block size	64 B
L2 cache	
	512KB, 1MB, 2MB or 4MB
ways	16 ways
latency	8 cycles
Block size	64 B
Memory	
latency	200 cycles

5.1 評価環境

フルシステムシミュレータ gem5[15] に SC と提案手法を実装し、これらの性能を評価した。なお、SC および提案手法は L2 キャッシュのみに実装し、L1 キャッシュのリプレースアルゴリズムには LRU 方式を用いた。この評価に用いたシミュレータ構成を表 1 に示す。

評価対象のプログラムとして、SPEC CPU2000 から 21 本を選択した。なお、プログラムの入力としては、SPEC CPU2000 の入力データの中で最もデータサイズの大きい ref データセットを用いて評価した。

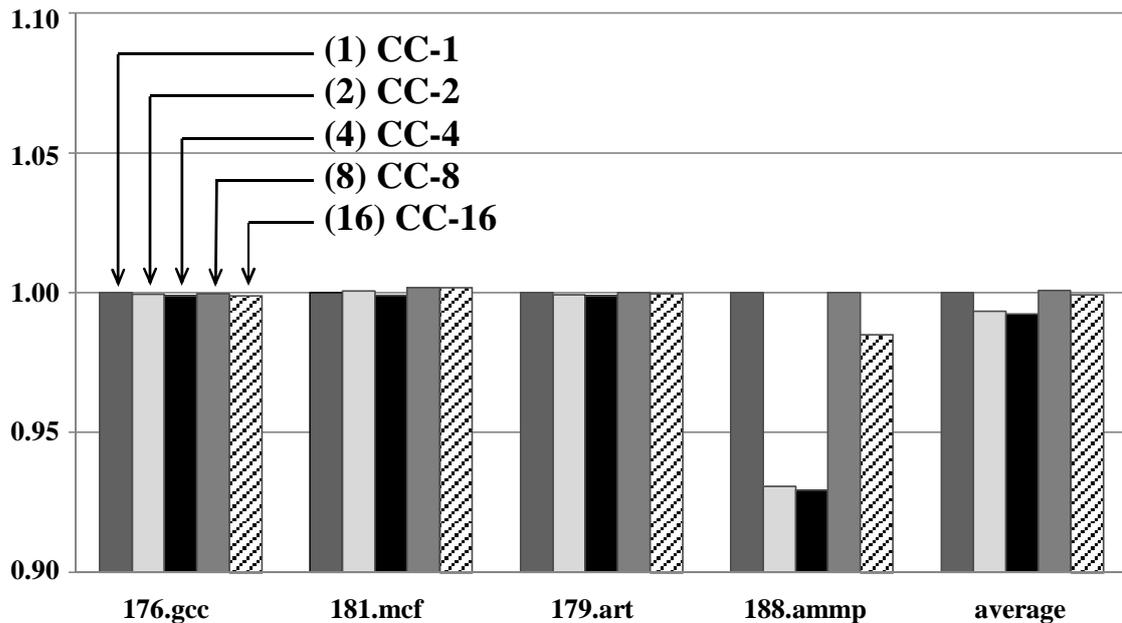


図 14: 予備評価の結果

5.2 Confidence Counter (CC) の設定

本提案手法では、ロード命令毎の特徴に基づく再参照間隔の予測値のうち、CC が閾値以上となったもののみをリプレース対象の選択に用いる。この閾値を設定するために、本研究の評価で用いるベンチマークである SPEC CPU2000 のうち SC でのキャッシュミス率が高い 176.gcc, 181.mcf, 179.art, 188.ammp に対する予備評価を行った。なお、SC のウェイ数は 4, L2 キャッシュのサイズは 512KB として、1G 命令スキップ後の 1G 命令を実行して評価した。

予備評価の結果を図 14 に示す。図中では、各ベンチマークプログラムの結果を 5 本のグラフで示しており、それぞれのグラフは左から順に、

- (1) CC の閾値を 1 としたモデル
- (2) CC の閾値を 2 としたモデル
- (4) CC の閾値を 4 としたモデル
- (8) CC の閾値を 8 としたモデル
- (16) CC の閾値を 16 としたモデル

のキャッシュミス率を表している。また、(1) のキャッシュミス率を 1 として正規化しており、その 0.9 以上の部分を示している。

評価結果から、188.ammp は (2), (4) での性能が良くなっているが、他のベンチマー

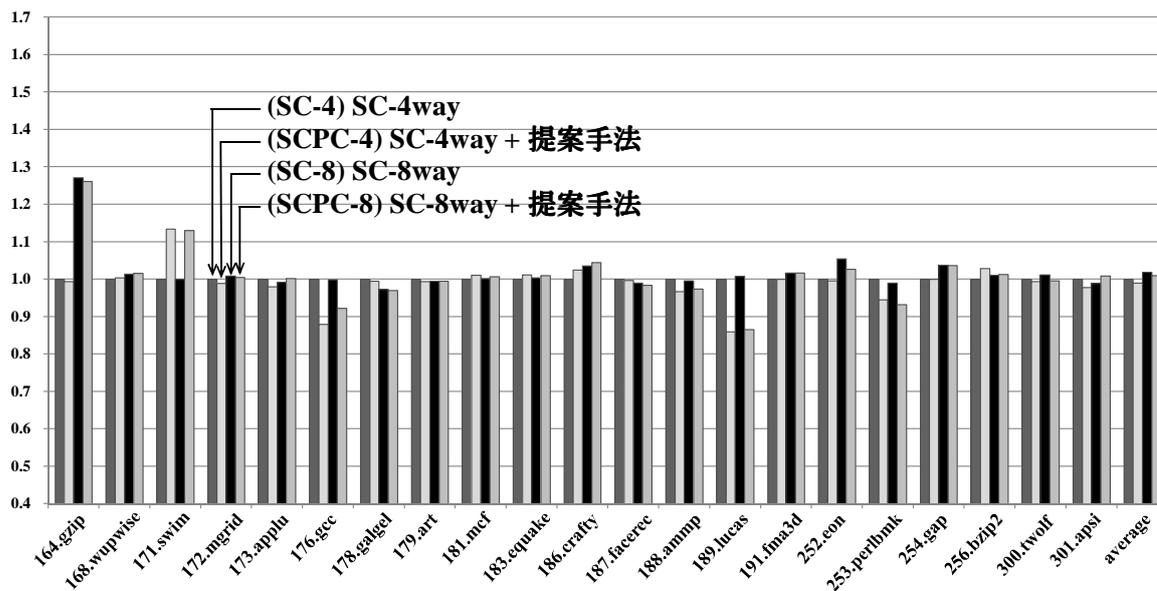


図 15: 512KB の L2 キャッシュにおける評価結果

クでは閾値を変えても性能に大きな変化はない。これは、一度でも再参照されたエントリは繰り返し参照される可能性が高かったためである。そこで本論文では、予備評価で最も性能の良い(4)の構成で評価を行う。

5.3 評価結果

予備評価の結果をもとに CC の閾値を 4 に設定し、2G 命令スキップ後の 3G 命令を実行して従来の SC および SC に提案手法を組み合わせたモデルについて評価を行った。図 15, 図 16, 図 17, 図 18 に、それぞれ L2 キャッシュサイズを 512KB, 1MB, 2MB, 4MB とした場合の評価結果を示す。また、図中では各ベンチマークプログラムの結果を 4 本のグラフで示しており、それぞれのグラフは凡例に示すように左から順に、

(SC-4) SC のウェイ数を 4 とした従来モデル

(SCPC-4) (SC-4) に提案手法を組み合わせたモデル

(SC-8) SC のウェイ数を 8 とした従来モデル

(SCPC-8) (SC-8) に提案手法を組み合わせたモデル

を利用して、各プログラムを実行した際のキャッシュミス率を示している。また、各グラフは従来モデル (SC-4) におけるキャッシュミス率を 1 として正規化しており、その値の 0.4 以上の部分を示している。

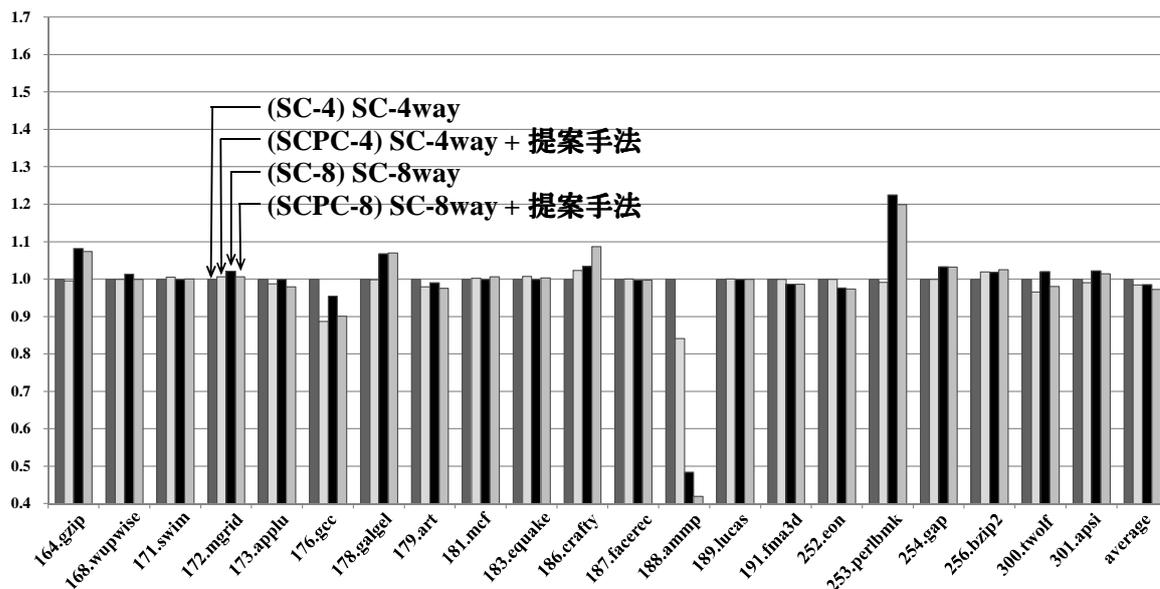


図 16: 1MB の L2 キャッシュにおける評価結果

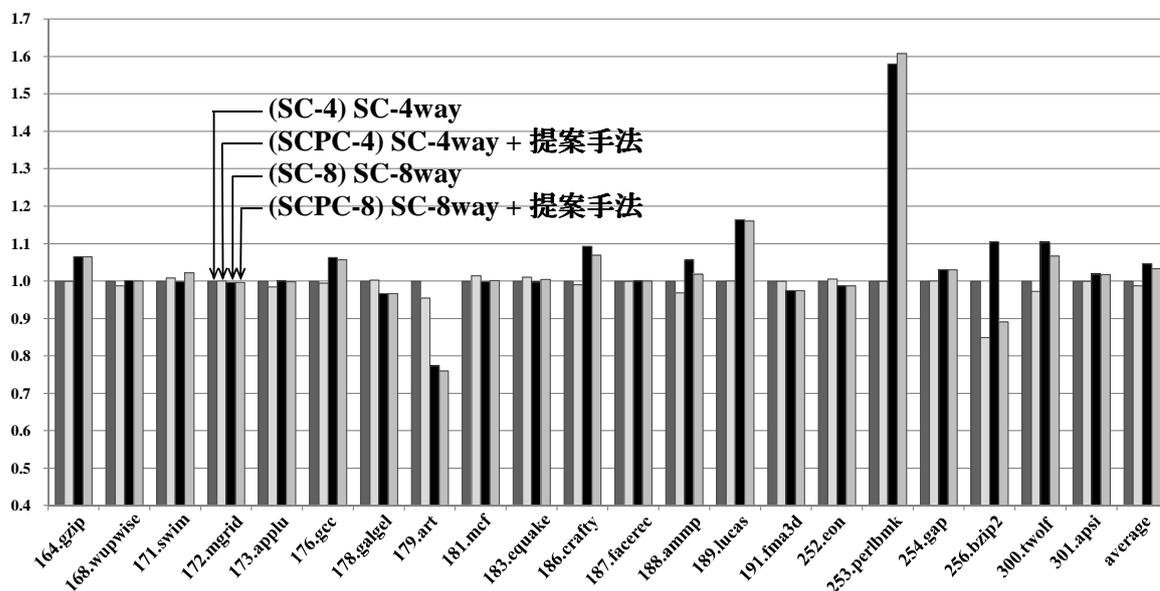


図 17: 2MB の L2 キャッシュにおける評価結果

また、本提案手法ではキャッシュヒット時にロード命令毎の特徴を収集するため、キャッシュサイズが変わることでキャッシュミス率が変化するプログラムでは、収集できるロード命令毎の特徴の量が変化すると考えられるため、キャッシュ構成によって提案モデルの性能が大きく変わる可能性がある。そこで、キャッシュミス率の変化に

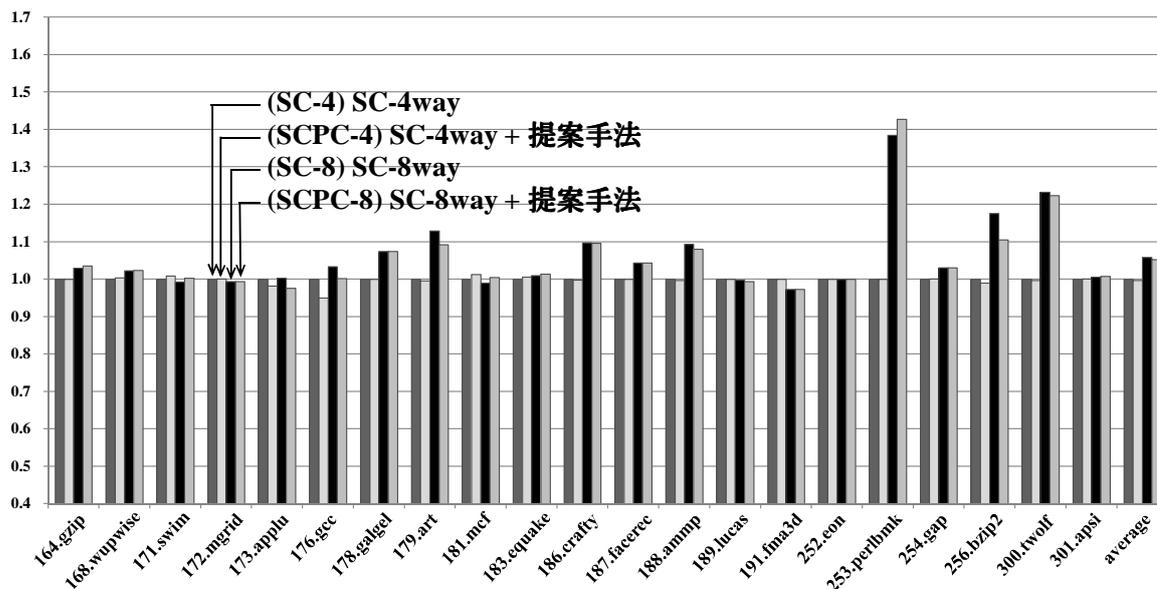


図 18: 4MB の L2 キャッシュにおける評価結果

よる提案モデルの性能の変化を検証するために、従来モデル (SC-4) において、キャッシュサイズを変えた場合のキャッシュミス率の変化を、図 19 に示す。それぞれのグラフは、凡例に示すように左から順に、キャッシュサイズを 512KB, 1MB, 2MB, 4MB とした場合のキャッシュミス率を示している。また、各グラフは 512KB 構成のキャッシュミス率を 1 として正規化している。

5.4 考察

まず、179.art と 256.bzip2 を見ると、キャッシュサイズが大きくなると、従来モデルに対して (SCPC-4), (SCPC-8) の両提案モデルの性能が向上している。なお、179.art は 1MB 構成で、256.bzip2 では 2MB 構成で最も性能向上率が高い。これらのプログラムはワーキングセットが大きいいため、キャッシュサイズが小さいとスラッシングが発生してしまう。しかし、図 19 に示すように、従来モデルではキャッシュサイズを大きくするとスラッシングの発生が抑えられ、キャッシュミス率が大幅に下がる。この結果、キャッシュヒット時に挿入される CM の値が増え、ロード命令毎の特徴も多く収集できるようになった。このため、ロード命令毎に同じ傾向を持つエントリがキャッシュに多く残ったと考えられる。また、171.swim も 179.art や 256.bzip2 と同様にワーキングセットが大きいプログラムである。しかし、このプログラムでは、キャッシュサイズ

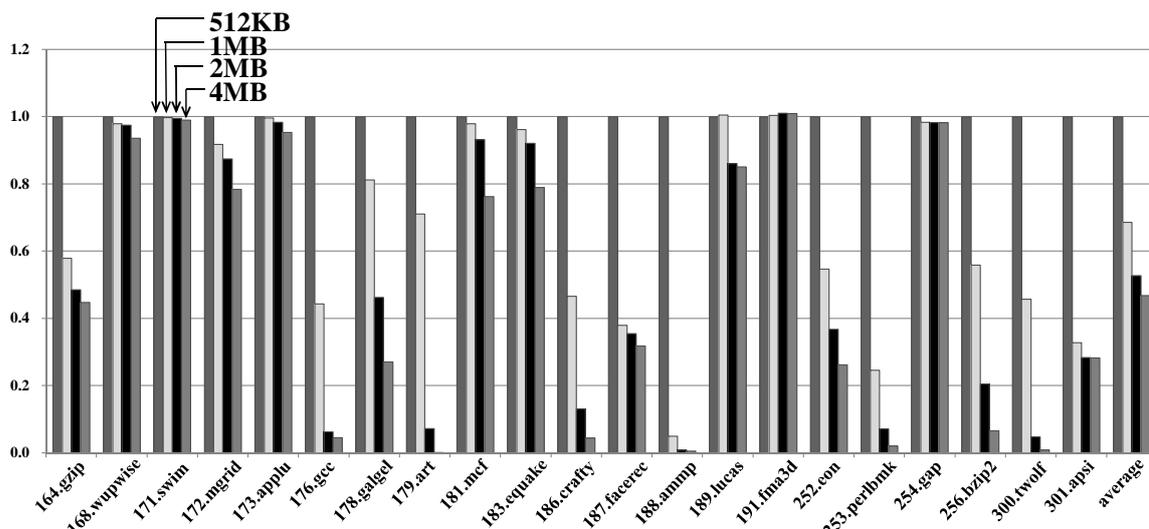


図 19: SC4 ウェイの従来モデルにおけるキャッシュミス率

512KB の構成において従来モデルに対して提案モデルの性能が低下しており、1MB 以上の構成では従来モデルと同程度の性能となっている。これは、図 19 に示すように、171.swim はキャッシュサイズが大きくなってもキャッシュミス率が下がらないプログラムであり、ロード命令毎の特徴を収集できないためである。

次に 188.ammmp を見ると、1MB 構成において、(SCPC-4) は (SC-4) に対して 15%、(SCPC-8) は (SC-8) に対して 14% の性能向上が得られている。188.ammmp は、文献 [10] によると LRU 方式を適用した場合の性能に対し、OPT を適用した場合の性能比が非常に高いプログラムである。また、SC でエントリ毎の特徴として収集している CM の値は、キャッシュリプレースを OPT に近づけるための情報である。このため、キャッシュサイズの増大に伴ってキャッシュミス率が下がるプログラムでは、ロード命令毎の特徴から予測される再参照間隔を用いたリプレースが増えることにより、キャッシュの動作が OPT に近づく。図 19 に示すように、188.ammmp は 512KB 構成に対して 1MB 構成のキャッシュミス率が大幅に低下している。このため、ロード命令毎の特徴を多く収集することができるようになり、ロード命令毎に同じ特徴を持つエントリがキャッシュに多く残ったと考えられる。なお、176.gcc、300.twolf も同様の理由で性能が向上した。

この一方で、文献 [10] では、172.mgrid、178.galgel、183.equake は OPT よりも LRU 方式を適用したキャッシュの方が性能が良く、これらのプログラムでは、OPT に近づけるほど性能が低下してしまうことが指摘されている。先述したように、本提案手法

表 2: 172.mgrid, 178.galgel, 183.quake のキャッシュミス率 (%)

	SC4 ウェイ				SC8 ウェイ			
	512KB	1MB	2MB	4MB	512KB	1MB	2MB	4MB
172.mgrid	4.54	4.17	3.97	3.56	4.58	4.26	3.96	3.53
178.galgel	8.61	7.00	3.98	2.33	8.38	7.48	3.85	2.50
183.quake	10.77	10.36	9.92	8.51	10.81	10.36	9.90	8.59

で収集している特徴はキャッシュリプレースの動作を OPT に近づけるための情報であるため、この情報を用いてリプレース対象を決定する機会が増えれば、LRU 方式に基づいたリプレースが適用される機会が減る。このため、これらのプログラムでは性能低下が予想されたが、評価結果では (SCPC-4), (SCPC-8) の両提案モデルで大幅な性能低下は見られなかった。ここで、これらのプログラムについて、従来モデルにおけるキャッシュミス率を表 2 に示す。この表から、これらのプログラムのミス率は最大でも 183.quake の 10% 程度であり、ロード命令毎の特徴を用いてリプレース対象を決定する機会も少ないと考えられる。また、キャッシュサイズが大きくなっても、キャッシュミス率の変化は最大でも 178.galgel の 6% 程度であるため、キャッシュミス率が低下することにより収集できるようになったロード命令毎の特徴は少ないと考えられる。このため、これらのプログラムでは提案手法によって大幅に性能が低下することはなかった。

次に、181.mcf を見ると、全てのキャッシュ構成において、従来モデルに対して提案モデルの性能が低下している。このプログラムは、ロード命令毎に特徴を持つエンタリが多いものの、その再参照間隔は非常に広い。また、ストリーミング処理や Scan が多発するため、再参照される可能性のあるエンタリは再参照される前に追い出されてしまい、エンタリ毎の再参照間隔予測が難しい。本提案手法では、ロード命令毎の特徴として CM の値を用いているが、この値はキャッシュ上のエンタリのヒット順を記録したものである。このため、SC で収集できる特徴が少ないことにより、ロード命令毎に収集できる特徴も減り、有用な特徴を利用してリプレース対象を決定できなかったと考えられる。

また、253.perlbnk にはスタックに対する操作が含まれており、3.1.1 節でも述べたように、ロード命令毎の特徴から過去の傾向と同じ間隔で再参照されると判断されている場合でも、過去の傾向とは異なる間隔で再参照されるエンタリが多いと考えられ

る。このような特徴をロード命令毎に抽出することは難しく、再参照間隔予測が外れてしまう可能性がある。この一方で、各エントリの特徴を利用して再参照間隔を予測する場合、そのエントリをキャッシュ上に配置したロード命令と再参照間隔との関係を考慮せずに、そのエントリの特徴のみを用いてリプレース対象を決定するため、スタックの操作に対しては、ロード命令毎の特徴を利用するよりも正確な予測ができると考えられる。このため、253.perlbnk では、全てのキャッシュ構成で従来モデルよりも提案モデルの性能が低下すると予想されたが、512KB 構成と 1MB 構成では提案モデルの性能は低下しなかった。これは、スタック操作以外の部分ではロード命令毎に特徴を持つエントリが存在し、キャッシュサイズが小さい時にはこのようなエントリに対してロード命令毎の再参照間隔予測が有効となるためである。しかし、キャッシュサイズが大きくなると、従来モデルにおいてもキャッシュミス率が低下するため、エントリ毎の特徴からも十分に再参照間隔予測ができるようになる。更に、上述したようにスタック操作に対してロード命令毎の特徴に基づく予測が外れるようになったため、2MB 構成と 4MB 構成において (SC-8) に対して (SCPC-8) の性能が低下したと考えられる。この一方で、186.crafty では、キャッシュサイズの小さい 512KB 構成と 1MB 構成において、従来モデルに対して提案モデルの性能が低下している。このプログラムは、特定のエントリにアクセスが集中するため、LFU 方式でのリプレースが有効だが、このような特徴はロード命令毎には収集できないため、提案モデルの性能が低下した。しかし、2MB 以上の構成ではキャッシュサイズが大きくなったため、アクセスの集中するエントリがリプレースされなくなり、性能低下が抑えられたと考えられる。

次に、173.applu, 191.fma3d および 254.gap を見ると、全てのキャッシュ構成において従来モデルと提案モデルが同等の性能を示している。これらのプログラムは、図 19 に示すように、(SC-4) でもキャッシュ構成が変わることによるキャッシュミス率の変化が小さい。更に、これらのプログラムでは、全てのキャッシュ構成において (SC-4) のキャッシュミス率は 10%未満である。このため、ロード命令毎の特徴に基づく再参照間隔予測はほとんど行われず、提案モデルでの性能向上は見られなかった。とで従来モデルのキャッシュミス率自体が低下し、4MB 構成の (SC-4) において、189.lucas で 16%, 252.eon で 0.01%となる。このため、キャッシュサイズが大きくなった場合に、191.fma3d など同様、従来モデルと提案モデルとの性能差が出にくくなったと考えられる。この一方で、168.wupwise, 187.facerec を見ると、191.fma3d などと同様に、全キャッシュ構成において従来モデルと提案モデルが同等の性能を示している。ここで、これらのプログラムについて、CC が閾値以上となる PCTable のラインの割合を調査

表 3: PCTable において CC が閾値以上となるラインの割合 (%)

	SC4 ウェイ				SC8 ウェイ			
	512KB	1MB	2MB	4MB	512KB	1MB	2MB	4MB
168.wupwise	1.57	7.92	7.95	7.81	1.79	8.21	8.18	8.52
187.facerec	4.29	8.99	10.20	10.69	5.42	10.85	13.20	9.91

した。結果を表 3 に示す。この結果から、これらのプログラムでは、CC が閾値以上となる予測値を持つラインは少なく、最大でも 187.facerec の 13% 程度であることがわかる。つまり、これらプログラムはロード命令毎の特徴が少なく、その予測値がリプレース対象の決定に利用されることはほとんどなかったため、提案モデルにおける性能向上は見られなかったと考えられる。最後に、164.gzip, 301.aspi は、ワーキングセットの小さいプログラムであるため、エン트리毎の特徴からでも十分に再参照間隔を予測でき、提案モデルにおける性能向上は見られなかった。

6 おわりに

本論文では、従来のキャッシュ性能向上手法をロード命令毎の特徴とエン트리毎の特徴という着眼点から分類して考察し、ロード命令およびエントリの特徴に基づく再参照間隔予測を用いたキャッシュリプレース最適化手法を提案した。

提案手法の有効性を確認するため、SPEC CPU 2000 を用いて評価を行った。この結果、エン트리毎の特徴のみを考慮した再参照間隔予測手法である既存の SC に対してキャッシュミス率を最大 19.4%、平均 1.1% 抑制することができた。これにより、提案手法の有効性を確認することができた。

本研究の今後の課題として、以下の二つが挙げられる。一つ目は、実装コストの削減である。本提案手法では、実行された全てのロード命令のプログラムカウンタを登録するため、PCTable の実装コストが非常に大きい。そこで、LRU 方式を用いて PCTable から有用ではないラインをリプレースしたり、定期的に PCTable をリセットする必要があると考えられる。

二つ目は、現在広く普及しているマルチコアプロセッサ構成においても有用となるモデルの提案である。マルチコアプロセッサでは、キャッシュを有効活用するために、複数のコアでラストレベルキャッシュを共有するケースが多い。このようなキャッシュ上では、単一コア構成よりも複雑なキャッシュリプレースが発生する。そのため、マル

チコア構成でも提案手法を評価し、有用な手法について今後検討していきたい。

謝辞

本研究のために、多大な御尽力を頂き、御指導を賜った名古屋工業大学の松尾啓志教授、津邑公暁准教授、齋藤彰一准教授、松井俊浩准教授、梶岡慎助教に深く感謝致します。また、本研究の際に多くの助言、協力をして頂いた松尾・津邑研究室、齋藤研究室および松井研究室の方々に感謝致します。特に、小田遼亮氏、澤田晃平氏、大平真司氏には研究を進めるにあたって多大な助言を頂きました。ここに深く感謝致します。

参考文献

- [1] Seznec, A.: A case for two-way skewed-associative caches, *Proc. 20th Annual Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 169–178 (1993).
- [2] Qureshi, M. K., Thompson, D. and Patt, Y. N.: The V-way Cache: Demand Based Associativity via Global Replacement, *Proc. 32nd Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 544–555 (2005).
- [3] D.Lee, J.Choi, J.H.Kim, S.H.Now, S.L.Min, Y.Cho and C.S.Kim: LRFU:A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies, *IEEE Trans. on Computers*, Vol. 50, No. 12, pp. 1352–1361 (2001).
- [4] Qureshi, M. K., Jaleel, A., Patt, Y. N., Steely, S. C. and Emer, J.: Adaptive insertion policies for high performance caching, *Proc. 34th Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 381–391 (2007).
- [5] Jaleel, A., Theobald, K. B., Steely, S. C. and Emer, J.: High performance cache replacement using re-reference interval prediction (RRIP), *Proc. 37th Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 60–71 (2010).
- [6] Khan, S. M., Wang, Z. and Jimenez, D. A.: Decoupled dynamic cache segmentation, *Proc. 18th IEEE Symp. on High Performance Computer Architecture (HPCA)*, IEEE Computer Society, pp. 1–12 (2012).
- [7] Collins, J. D., Z, H. W., Tullsen, D. M., Hughes, C., fong Lee, Y., Lavery, D. and Shen, J. P.: Speculative Precomputation: Long-range Prefetching of Delinquent-Loads, *Proc. 28th Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 14–25 (2001).

- [8] Lai, A.-C., Fide, C. and Falsafi, B.: Dead-block prediction and dead-block correlating prefetchers, *Proc. 28th Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 144–154 (2001).
- [9] Belady, L. A.: A study of replacement algorithms for a virtual-storage computer, *IBM Systems Journal*, Vol. 5, No. 2, pp. 78–101 (1966).
- [10] Rajan, K. and Govindarajan, R.: Emulating Optimal Replacement with a Shepherd Cache, *Proc. of 40th IEEE/ARM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 445–454 (2007).
- [11] Bansal, S. and Modha, D. S.: CAR: Clock with Adaptive Replacement, *the 3rd USENIX Conference on File and Storage Technologies*, USENIX, pp. 187 – 200 (2004).
- [12] 堀部悠平, 三輪忍, 塩谷亮太, 五島正裕, 中條拓伯: 選択的キャッシュ・アロケーション: マルチスレッド環境におけるキャッシュ利用効率の向上手法, *情報処理学会研究報告 2010-ARC-190*, Vol. , No. 1, pp. 1–8 (2010).
- [13] Lai, A.-C. and Falsafi, B.: Selective, Accurate and Timely Self-Invalidation Using Last-Touch Prediction, *Proc. 27th Annual Int'l Symp. on Computer Architecture (ISCA)*, ACM, pp. 139–148 (2000).
- [14] Khan, S. M., Tian, Y. and Jimenez, D. A.: Sampling Dead Block Prediction for Last-Level Caches, *Proc. 43rd Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, IEEE Computer Society, pp. 175–186 (2010).
- [15] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen5, R., Sewell, K., Shoab, M., Vaish, N., Hill, M. D. and Wood., D. A.: The gem5 Simulator, *AMC SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7 (2011).