修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院情報システム学研究科　情報ネットワークシステム学専攻 博士前期課程		
氏　　名	眞島　一貴	学籍番号	1152034
論 文 題 目	プリフェッチ情報を利用したキャッシュ置き換え手法の検討		

要　旨

計算機の性能向上のボトルネックに，プロセッサとメモリ間のアクセスレイテンシが挙げられる．このアクセスレイテンシを隠蔽するために，プロセッサの近くに小容量で高速なアクセスができるキャッシュメモリを置き，キャッシュを介したデータや命令の転送が行われている．しかし，キャッシュは小容量であるため，必要なデータを格納できないことがあり，性能の低下を招いている．

キャッシュを無駄なく使うためには，キャッシュラインにどのようなデータを載せるか，あるいは追い出すかが重要である．キャッシュを管理するキャッシュアルゴリズムには，LRU(Least Recently Used) が広く使われているが，プログラムによってはLRU を用いることで性能が低下する問題が指摘されており，RRIP(Re-Reference Interval Prediction) をはじめ，問題を解決するためのアルゴリズムが数多く提案されている．またプリフェッチを用い，キャッシュラインに予めデータを入れることでキャッシュミスを防ぐ手法も多く提案されている．しかし，プリフェッチは実行するプログラムやキャッシュサイズの大小によって効果が変化することから，性能が低下してしまうことが指摘されている．

本研究では，高効率なキャッシュ性能を実現するために，データの置き換えとプリフェッチを組み合わせ，局所性のあるデータの追い出し防止を目的としたキャッシュアルゴリズムを提案する．プリフェッチによって得られたデータに，キャッシュライン上に格納されているデータが含まれていたならば，近い未来にアクセスされるデータをキャッシュラインに残すようにキャッシュの追い出し順序を変更することで，キャッシュの処理性能を向上させる．シミュレータを用いた性能評価を行った結果，プリフェッチを行った既存手法に対し，キャッシュ性能を向上させることができた．

電気通信大学大学院情報システム学研究科

2013 Jan.

修　士　論　文

プリフェッチ情報を利用した
キャッシュ置き換え手法の検討

指導教員　入江 英嗣 准教授

副指導教員　吉永 努 教授

副指導教員　長岡 浩司 教授

平成 25 年 1 月 24 日

提 出 者

所 属　　大学院情報システム学研究科

情報ネットワークシステム学専攻

学籍番号　　1152034

氏 名　　眞島 一貴

(表紙裏)

目 次

図 目 次

表 目 次

第1章　序論

計算機の性能向上のボトルネックに，プロセッサとメモリ間のアクセスレイテンシが挙げられる．このアクセスレイテンシを隠蔽するために，プロセッサの近くに小容量で高速なアクセスができるキャッシュメモリを置き，キャッシュを介したデータや命令の転送が行われている．しかし，キャッシュは小容量であるため，必要なデータを格納できないことがあり，性能の低下を招いている．

キャッシュを無駄なく使うためには，キャッシュラインにどのようなデータを載せるか，あるいは追い出すかが重要である．キャッシュを管理するキャッシュアルゴリズムには，LRU(Least Recently Used) が広く使われているが，プログラムによっては LRU を用いることで性能が低下する問題が指摘されており，RRIP(Re-Reference Interval Prediction) をはじめ，問題を解決するためのアルゴリズムが数多く提案されている．また，キャッシュラインに予めデータを入れることでキャッシュミスを防ぐプリフェッチの手法についても多く提案されている．しかし，プリフェッチは実行するプログラムやキャッシュサイズの大小によって効果が変化することから，性能が低下してしまうことが指摘されている．

本研究では，高効率なキャッシュ性能実現のため，ライン置き換えとプリフェッチを組み合わせ，局所性のあるデータの追い出し防止を目的としたキャッシュアルゴリズムを提案する．プリフェッチによって得られたデータに，キャッシュライン上に格納されているデータが含まれていたならば，近い未来にアクセスされるデータをキャッシュラインに残すようにキャッシュの追い出し順序を変更することで，キャッシュの処理性能を向上させる．シミュレータを用いた性能評価を行った結果，プリフェッチを行った既存手法に対し，最大で 42%，幾何平均で 2.2% の性能向上が得られた．

以降，本論文は次のように構成される．第 2 章ではキャッシュメモリについて述べ，第 3 章で既存のキャッシュアルゴリズムの研究について述べる．第 4 章で提案するプリフェッチを用いた置き換え手法を述べ，第 5 章でその実装について述べる．第 6 章では評価環境を示す．第 7 章でプロセッサシミュレータを用いた評価・比較を行い，第 8 章でまとめる．

第2章　キャッシュメモリ

2.1　キャッシュメモリ

キャッシュメモリ (Cache Memory) とは，CPU の近くに置かれる，高速で小容量のメモリのことである．プログラムには，最近利用したデータや命令を再利用する局所性がある．この局所性を用いることで，最近のアクセスから，プログラムが次にどのようなデータや命令を利用するのかを予測することができる．キャッシュはこれを利用したもので，アクセスに時間のかかるメモリと CPU との間のデータ転送の代わりに，キャッシュにアクセスするデータを格納し，CPU とデータ転送を行うことで，メモリと CPU との間のアクセスレイテンシを隠蔽し，データ転送効率を向上させている．一般にキャッシュは，CPU に近い方から，高速で小容量な 1 次キャッシュ，やや高速で容量がやや大きい 2 次キャッシュなどと，複数段のキャッシュを並べて用いられる．

しかし，キャッシュによる恩恵が受けられるのは，命令などアクセスされるデータがキャッシュ内に保持されている場合だけである．キャッシュは小容量であること，また一度アクセスされないとデータが保持できないため，次のようなミスが発生する．

1. 初期参照ミス
   キャッシュへ読み込まれていないブロックに最初にアクセスした時に発生する

2. 容量性ミス
   キャッシュの容量が小さくプログラムの実行に必要となる全てのデータがキャッシュ内に保持できない

3. 競合性ミス
   キャッシュのブロックが複数ブロックが同じセットのデータを取り合う

この対策として，キャッシュの容量やウェイ数を大きくしたりするほか，複数段のキャッシュを使用することが対策として挙げられる．しかし，キャッシュ容量を大きくすると，容量性ミスを減らせる一方でアクセス時間が増加するように，対策による悪影響も起こってしまう

2.2　キャッシュへのアクセス

キャッシュへのアクセスには，最近アクセスされたアドレスは近い未来にアクセスされやすい時間的局所性と，最近アクセスされたアドレスの周囲は近い未来にアクセスされやすい空間的局所性を持っている．キャッシュへのメモリ参照には，次の 4 つがよくあるアクセスとして挙げられる．

1. Recently-friendly Access Patterns
   参照に局所性がある．キャッシュの機能を生かせる

2. Thrashing Access Patterns
   キャッシュブロックサイズよりデータサイズの大きく，キャッシュがヒットしなくなる．また，キャッシュの中身が追い出されてヒットしなくなる．スラッシングでは，ブロックサイズが大きい場合でも，キャッシュ内データを保持できることが好ましい

3. Streaming Access Patterns
   一度しか使用されないデータの参照が連続し（これをスキャンという），キャッシュから局所性のあるデータが追い出され，役にたたないデータで埋まってしまう

4. Mixed Access Patterns
   複数のアクセスパターンが混合したパターン．スキャンが含まれていると局所性のあるデータがキャッシュから追い出される．スキャンがあった場合でも，頻繁に使われるデータが保持されていることが好ましい

2.3 キャッシュの改良

キャッシュ性能を向上させる方法として，次の改良法が必要とされている [1, 2].

1. ミス率の削減
   キャッシュミスが発生すると，メモリへのデータを読み込みに行くペナルティが発生してしまう．キャッシュライズやキャッシュサイズを大きくする他，ウェイ数を増やすことで改善できる

2. ミスペナルティの削減
   キャッシュを複数階層並べることで，キャッシュミスによるペナルティを小さくすることができる．また，書き込みに対して読み出しを優先的に行うことで改善できる

3. ヒット時の時間削減
   キャッシュをインデックスする際の仮想アドレスから物理アドレスへの変換を省略することで改善できる

第3章　キャッシュの性能向上に向けた研究

3.1　ライン置き換えアルゴリズム

3.1.1　LRU

キャッシュアルゴリズムは，次の 3 種類に大別される．

1. キャッシュに置き換えに着目した手法
2. プリフェッチに着目した手法
3. 置き換えとプリフェッチを組み合わせた手法

キャッシュの置き換えアルゴリズムには，ランダムに選ぶ方法，キャッシュに入れられた順に追い出していく FIFO(First In First Out) の他，最近最も使われていないデータをキャッシュから追い出す LRU が代表的なものとして挙げられ，一般には参照の局所性による恩恵を受けられる LRU が使われている．

LRU は，キャッシュ上で保持しているデータが使用された順序を記憶し，データを最近使用された (Most Recently Used: MRU) ものから順に並べ，使用されていないデータを LRU 側に移動させ，最も最近使用されていないデータをキャッシュから追い出すアルゴリズムである．図 3.1.1 は LRU の概要である．LRU では次の手順で置き換えを行う．

1. キャッシュラインに初めて入ってくるデータはキャッシュの MRU に載る
2. キャッシュライン上で格納されている間にデータが参照された場合，データは MRU に移動する
3. キャッシュライン上でデータが保持しきれなくなり，追い出しが必要になった場合は LRU にあるデータを追い出す

LRU は参照に局所性あるデータをキャッシュに格納することでき，アクセスレイテンシを隠蔽できることから，現在まで広く使われている．他方，LRU により性能が向上できないことが指摘されている．これらのアクセスパターンのうち，LRU では Recently-friendly Access Patterns 以外のアクセスにより，キャッシュ性能決定に重要な局所性のあるデータがキャッシュから追い出されてしまうため，キャッシュ性能が低下している．このため，これらの弱点を克服したより高性能なキャッシュアルゴリズムの提案が求められている．

3.1.2　LRU に代わる置き換えアルゴリズム

ライン置き換えアルゴリズムは，LRU が持つ弱点を克服し，局所性を持つデータをキャッシュに残せるようにするための手法が数多く提案されている．

Neil らは，直近の参照だけではなく，その前の参照時も利用して置き換えを行う LRU-k を提案した [3]．Lee らは参照された時間から置き換えを行う LRU と参照された頻度から置き換えを行う LFU(Least Frequency Used) を組み合わせ，スキャンに耐性を持たせる手法を提案した [4]．スキャン耐性を持つアルゴリズムには，他に Johnson ら [5] や，Megiddo ら [6] のように，キャッシュにあり再参照されたデータを格納するキュー，キャッシュにあるが再参照されていないデータを格納するキュー，あるいはキャッシュから追い出されたデータを格納するキューなどと複数のキューを持たせ，それらを組み合わせて置き換えを行う手法も提案されている．

Qureshi らは，スラッシングやスキャンによるミス低減のため，キャッシュの挿入位置を変更する手法 (LIP) と共に，キャッシュへのアクセス状況を測定し，LRU と LIP から適切な手法を動的に切り替える手法を提案した [7]．Jaeel らは， [7] を共有キャッシュでも使用できるように拡張した [8]．また Jaleel らは，擬似 LRU である NRU を基に各データにいつ再び参照されるかを意味する情報を持たせ，キャッシュラインでの挿入時と再参照時との処理を区別して置き換えを行う手法を提案した [9]．Michaud らは異なる 2 つ以上の置き換え手法を組み合わせ，動的に切り替える手法を提案した [10]．Wu らは，追い出されたラインが再参照されたかどうかを用いてキャッシュの再参照パターンを予測する手法を提案した [11]．Seshadri らは，最近追い出されたブロックのアドレスを追跡し，スラッシングの影響を軽減する手法を提案した [12].

3.1.3 RRIP

Jaeel らは，スキャンやスラッシングが LRU の性能に悪影響を与えていることから，スキャンに耐性を持つアルゴリズム SRRIP とスラッシングに耐性を持つアルゴリズム BRRIP，また SDM(Set Dueling Monitor) を用いて SRRIP と BRRIP とでミスが少ない方のアルゴリズムを動的に切り替えて使用するアルゴリズム DRRIP を提案した．図 3.1.2 はその概要である．RRIP では，LRU の MRU 側，LRU 側といった置き換え順序を判断する指標として，再参照されるまでにかかる時間を意味する RRPV(Re-reference Prediction Values) を用いている．RRPV は各ブロックごとに M ビットの値を持っており，RRPV が 0 だと近いうちに再参照されることを示し，RRPV が $2^M - 1$ だと再参照まで時間がかかることを示している．SRRIP では次の手順で置き換えを行う．図 3.1.3 は置き換えアルゴリズム図である．なお，以下では文献に従い RRPV を 2 ビットとした．

キャッシュヒット時

1. ヒットしたブロックの RRPV を 0 にする

キャッシュミス時

1. RRPV=3 のブロックを探索する
2. RRPV=3 のブロックが見つかった場合はそのブロックを置き換える
3. いずれのブロックも見つからなかった場合は，全ての RRPV を +1 して，最初からやり直す
4. 新たに入ってきたデータの RRPV を 2 にする

同様に，BRRIP では次の手順で置き換えを行う．

キャッシュヒット時

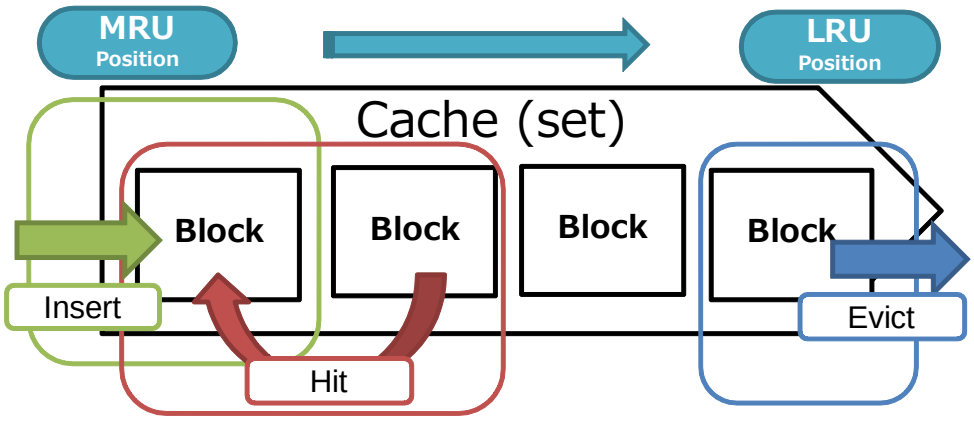

図 3.1.1: LRU の概要

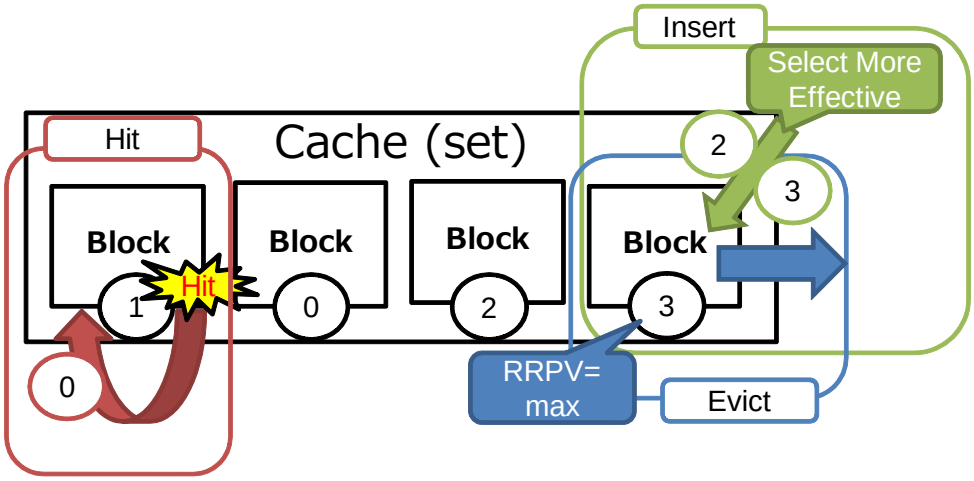

図 3.1.2: RRIP の概要

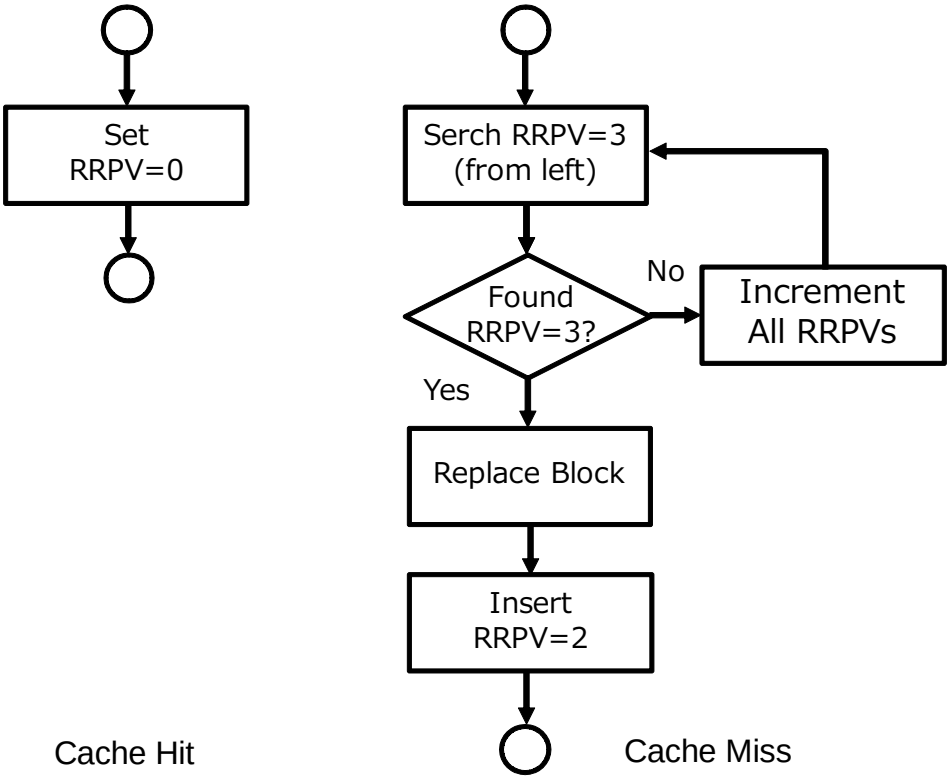

図 3.1.3: SRRIP での置き換えアルゴリズム

1. ヒットしたブロックの RRPV を 0 にする

キャッシュミス時

1. 空のブロックもしくは RRPV=3 のブロックを探索する
2. RRPV=3 のブロックが見つかった場合はそのブロックを置き換える
3. いずれのブロックも見つからなかった場合は，全ての RRPV を +1 して，最初からやり直す
4. 新たに入ってきたデータの RRPV を 3 にする．ただし，確率 ϵ で RRPV を 2 にする（[9] では $\epsilon = 1/32$ としている）．

また DRRIP では，キャッシュラインに入っているデータを 32 セットに分け，2 つの SDM として，残りを Follower set とした上で，SRRIP と BRRIP を次のように切り替えて使用する．

1. SDM のうち片方では SRRIP を使用し，もう一方では BRRIP を使用しミスした数を測定する．
2. ミスした数は PESL と呼ばれるカウンタを用いて測定し，SRRIP でミスした場合は PESL を+1 し，BRRIP でミスした場合は-1 する
3. Follower set では PESL を参照し，ミスが少なかった方（PESL が 0 以下であれば RRIP，そうでなければ BRRIP）のアルゴリズムを使用する

3.1.4 デッドブロック予測アルゴリズム

デッドブロック (Dead Block) とは，キャッシュから追い出されるまで二度と再参照されないデータのことをいう．キャッシュに挿入されたデータは，何度か再参照される（この状態をライブブロックという）が，その後デッドブロックとなり再参照されないままキャッシュに残ることから，キャッシュの利用効率を低下させてしまう．キャッシュの効率化のためには，最後に参照されるタイミングを予測し，早く追い出す必要がある．

ライブブロックである間の挙動からデッドブロックの予測を行う手法は複数提案されており，Lai らは命令カウンタの参照回数から予測する手法を [13]，Kharbutli らはブロックの参照回数から予測する手法を [14]，Hu らはデータが L1 キャッシュに残っている時間から予測する手法を提案した [15]．また，Abella らは再度参照されるまでとデッドブロックになるまでとの時間の相関関係から予測する手法を提案した [16]．Liu らは L1 キャッシュでのアクセスが短時間に同じブロックに集中する傾向があることに着目し，キャッシュ内に存在する二度と参照されないデータの発生を予測し，追い出す手法を提案した [17].

これらデッドブロック予測アルゴリズムでは，置き換えやプリフェッチと組み合わせた手法も提案されている．

3.2 プリフェッチアルゴリズム

プリフェッチにより，予めこれからアクセスされるデータをキャッシュに転送することで性能を向上させる手法が提案されている．プリフェッチには，CPU がアクセスしたアドレスから次のアドレスも近い将来使われると予測し，データを予めキャッシュに挿入する手法と，プリフェッチ専

用の命令により次に使われるアドレスを予測してデータを予めキャッシュに挿入する手法がある．これらの手法によりキャッシュミスを防ぐことで，キャッシュの性能を向上させることができる．実行するプログラムやキャッシュサイズの大小によって効果が変化することから，プリフェッチによりキャッシュにある既存のデータが追い出され，性能が低下してしまう場合あり，課題となっている．

Wang らは，大きなキャッシュでもメモリアクセスレイテンシにより性能が低下していることに加え，プリフェッチの効果も部分的で無駄になっていることに着目し，L2 キャッシュミス時にチャネルを追加する手法を提案した [18]．堀部らはプログラム実行時の事象の多くが実行パスに依存していることに着目し，実行パスを考慮した手法を提案した [19]．Ebrahimi らは，プリフェッチするアドレスかどうかをハードウェアで通知する機構を用意し，また実行時から得られるフィードバックを用いてプリフェッチ量を制御する手法を提案した [20]．入江らは，予め設定した閾値を基準に用いたプリフェッチ量の制御ではプログラムによって適切な制御ができないことに着目し，LRU と MRU とのデータの対流に着目してプリフェッチ量を制御する手法を提案した [21]．

3.3 組み合わせアルゴリズム

3.3.1 置き換えとプリフェッチを組み合わせたアルゴリズム

置き換えとプリフェッチを組み合わせたキャッシュアルゴリズムが提案されている．

石井らはプリフェッチが予測したメモリアクセスパターンが時間的局所性に乏しいことを利用し，プリフェッチで得られたデータを積極的にキャッシュから追い出す手法を提案した [22]．Zhang らは，置き換えだけでは性能向上が難しいとして，Pseudo-LRU によるキャッシュのライン置き換えとプリフェッチを組み合わせた手法を提案した [23]．Wu らは，文献 [9] をもとにプリフェッチによりポリューションに着目し，キャッシュのヒットやミスに基づいてプリフェッチ時におけるキャッシュの挿入位置を変える手法を提案した [24]．

3.3.2 PACMan

Wu らは，プリフェッチによってキャッシュに入れられたラインが使われにくい点に着目し，キャッシュへのアクセスを Demand によるアクセスと Prefetch によるアクセスに切り分け，以下のようなラインの挿入方法を提案した，図 3.3.1 から図 3.3.4 に各手法の概要を示す．文献 [24] では，これらの手法を RRIP に対して適用して評価を行った．

1. キャッシュでのプリフェッチのリクエストがミスした時から再参照を予測するため，プリフェッチによってキャッシュラインに入ってくるデータを LRU 側に配置する手法 (PACMan-M)

2. キャッシュでのプリフェッチのリクエストがヒットした時を用いて再参照を予測するため，プリフェッチによってヒットしたキャッシュラインを MRU 側に移動させない手法 (PACMan-H)

3. キャッシュでのプリフェッチリクエストがミスした時とヒットした時の両方を用いて再参照を予測するため，プリフェッチによってキャッシュラインに入ってくるデータを LRU 側に配置し．またプリフェッチによってヒットしたキャッシュラインを MRU 側に移動させない手法 (PACMan-HM)

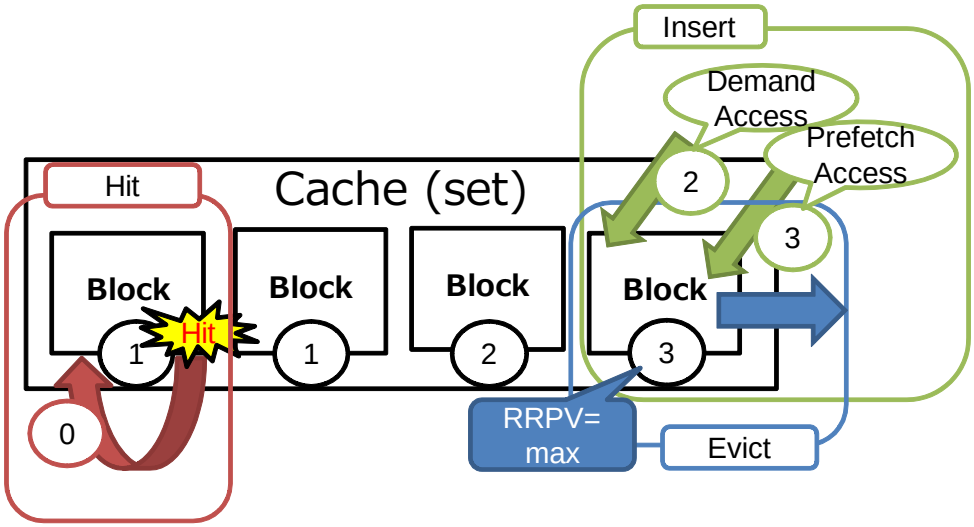

図 3.3.1: PACMan-M の概要

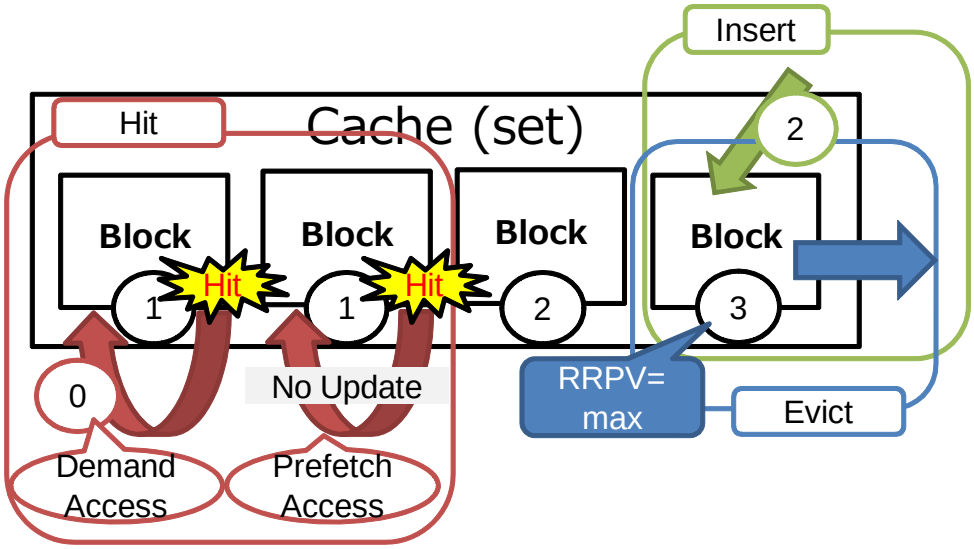

図 3.3.2: PACMan-H の概要

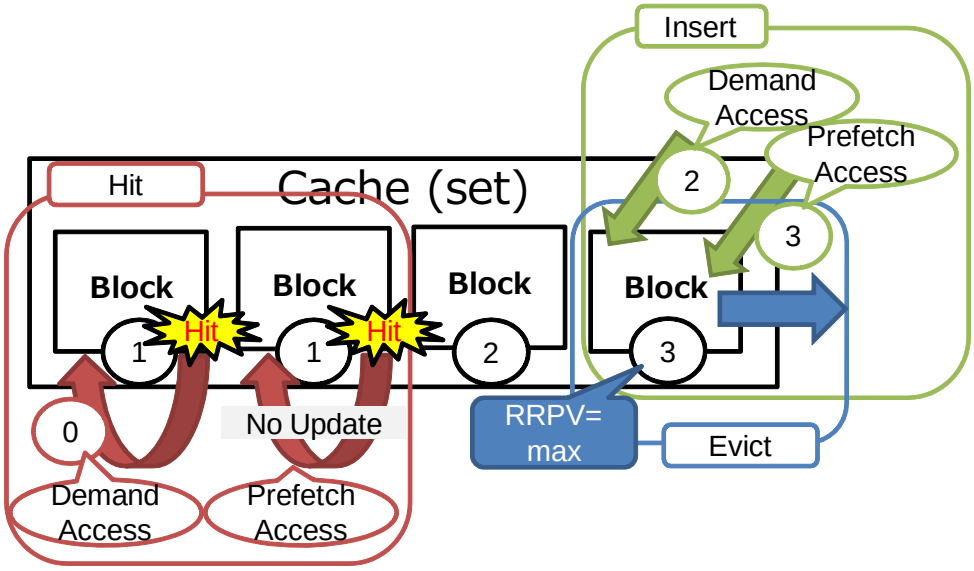

図 3.3.3: PACMan-HM の概要

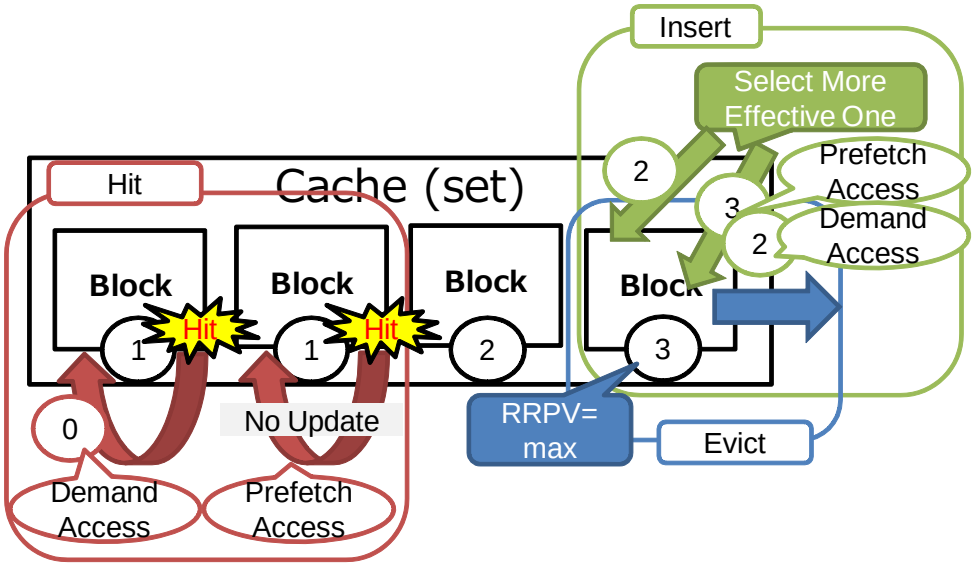

図 3.3.4: PACMan-DYN の概要

表 3.1: RRIP に適用した PACMan での RRPV 挿入アルゴリズム

	PACMan-M		PACMan-H		PACMan-HM	
SRRIP	Demand	Prefetch	Demand	Prefetch	Demand	Prefetch
Insertion	2	3	2	2	2	3
Re-Reference	0	0	0	No Update	0	No Update
BRRIP	Demand	Prefetch	Demand	Prefetch	Demand	Prefetch
Insertion	Mostly 3	Mostly 3	Mostly 3	Mostly 3	Mostly 3	Mostly 3
Re-Reference	0	0	0	No Update	0	No Update

4. PACMan-H を SRRIP に適用した場合，PACMan-HM を SRRIP に適用した場合，また PACMan-H を BRRIP に適用した場合のそれぞれについてキャッシュミス数を測定し，最もミスが少なかったアルゴリズムを用いることでプリフェッチの利益を得やすくする手法 (PACMan-DYN)

表 3.1 は PACMan-M, PACMan-H, PACMan-HM について，SRRIP 及び BRRIP に適用した場合での RRPV の挿入位置を示したものである．また PACMan-DYN では，3 つの SDM を使用し，SRRIP を適用した PACMan-H，SRRIP を適用した PACMan-HM，また BRRIP を適用した PACMan-H を BRRIP と各々に対応したカウンタを用いて，次のように使用するアルゴリズムを決定している．

1. PACMan-H を SRRIP に適用した SDM でミスした場合，自身に対応するカウンタに 2 を足し，他のカウンタから 1 を引く
2. PACMan-HM を SRRIP に適用した SDM でミスした場合，自身に対応するカウンタに 2 を足し，他のカウンタから 1 を引く
3. PACMan-H を BRRIP に適用した SDM でミスした場合，自身に対応するカウンタに 2 を足し，他のカウンタから 1 を引く
4. Follower sets では 3 つのカウンタから最も値の小さい値を持つカウンタに対応するアルゴリズムを用いる

しかし，この手法はプリフェッチによってキャッシュラインに入れられるデータに着目し，プリフェッチで入れられたデータが再参照されないために追い出されやすくなるように提案したものであり，現在キャッシュラインに格納されているデータに着目した検討はなされていない．

第4章　プリフェッチを用いたキャッシュの置き換え

4.1　RRIPの問題点

第 2 章で述べたように，LRU よりも効率的なライン置き換えアルゴリズムが提案されている．またプリフェッチも小型の携帯端末をはじめ，多くの計算機で広く使われている．しかし，ライン置換えアルゴリズムとプリフェッチを組み合わせて用いると，キャッシュ性能が低下する可能性がある．

図 4.1.1 は，ベンチマークとして SPECCPU2006INT 及び SPECCPU2006FP を使用し，L2 キャッシュで 100%ヒットする場合に対して，L2 キャッシュにそれぞれ LRU, SRRIP, BRRIP, DRRIP を適用した時のクロックあたりの実行命令数 (IPC) の幾何平均を，L2 キャッシュサイズを 64KB から 4MB まで変化させて比較したものである．また，図 4.1.2 は，プリフェッチを行った場合，L2 キャッシュに LRU, SRRIP, BRRIP, DRRIP をそれぞれ適用した場合の IPC を同様に取ったものである．プリフェッチを行なわない場合では，RRIP の IPC は LRU の IPC と同じか上回っているが，プリフェッチを行った場合では IPC ではキャッシュサイズにより逆に LRU が RRIP を上回っており，特に BRRIP，DRRIP ではその差が大きくなっている．これは RRIP ではプリフェッチによる性能向上が得られないベンチマークが存在するためである．

SPECCPU2006 から特徴的なベンチマークを選ぶ．図 4.1.3，図 4.1.5 は，SPECCPU2006 から 456.hmmer と 433.milc を選びプリフェッチを行わなかった場合の IPC である．また図 4.1.4，図 4.1.6 は，プリフェッチを行った場合の IPC である． hmmer では，プリフェッチを行なわなかった場合，BRRIP 以外が同じ程度の IPC となるのに対し，プリフェッチを行った場合では SRRIP では最大で 10%，BRRIP では 25%，DRRIP でも 20%性能が低下している．特に L2 キャッシュサイズが小さい場合では，プリフェッチを行わなかった場合よりも IPC が低下している．

一方 milc は，プリフェッチを行うことで性能が向上できるベンチマークであるが，LRU では，容量が小さい場合でもプリフェッチの恩恵が受けて IPC が向上しているのに対し，RRIP では性能が LRU ほど上がらず，特に BRRIP や DRRIP ではキャッシュサイズが 1MB を超えないと IPC が向上していない．

キャッシュに入るアクセスについて，アドレスと時間との関係を確認する．図 4.1.7，図 4.1.8 は，それぞれ SPECCPU2006FP から 433.milc を選び，プリフェッチを行い，L2 キャッシュにそれぞれ LRU,SRRIP を使用した場合に，L2 キャッシュに入ってくるアクセスのアドレスと時間との関係を示したものである．また，図 4.1.9，図 4.1.10 は，図 4.1.7，図 4.1.8 を拡大したものである． どちらのアルゴリズムでも時刻経過とアドレス移動が正比例の関係であるが，図 4.1.8 の傾きは図 4.1.7 のそれと比べ，傾斜が緩くなっており，clock が 410000000 付近からさらに傾きが緩やかになっていることが確認できる．ここからキャッシュの処理性能が低下していると言える．また，図 4.1.9，図 4.1.10 を見ると，近くのアドレスに対して複数のアクセスがあることが確認できる．別のベンチマークを用いてアクセスの状況を確認する．図 4.1.11，図 4.1.12 はそれぞれに LRU,SRRIP を使用

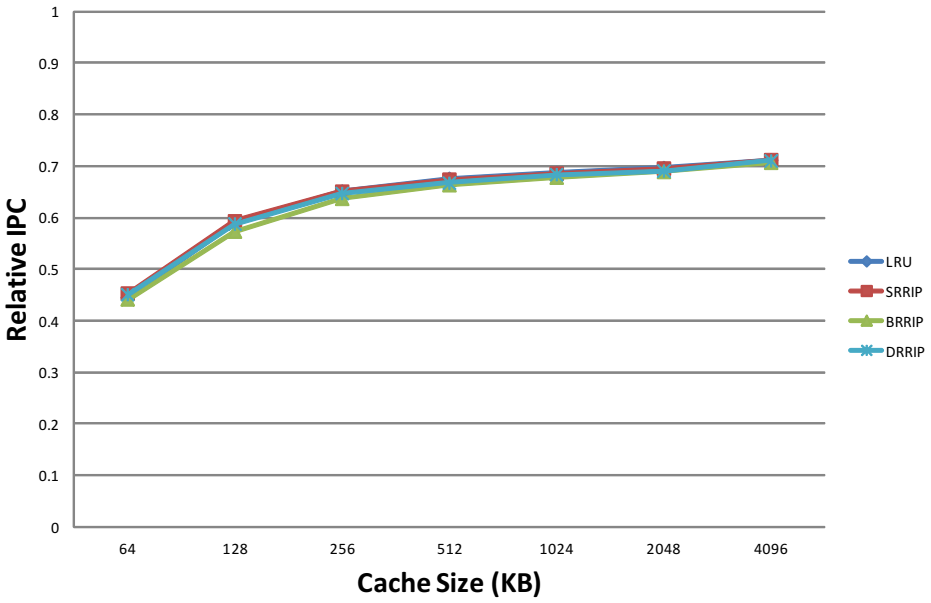

図 4.1.1: プリフェッチを行わない場合の IPC の比較

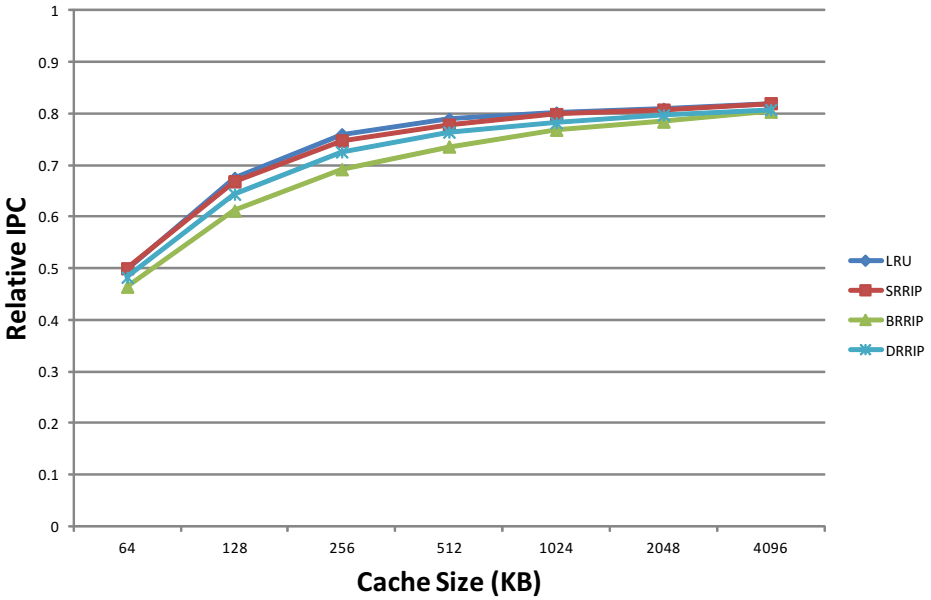

図 4.1.2: プリフェッチを行った場合の IPC の比較

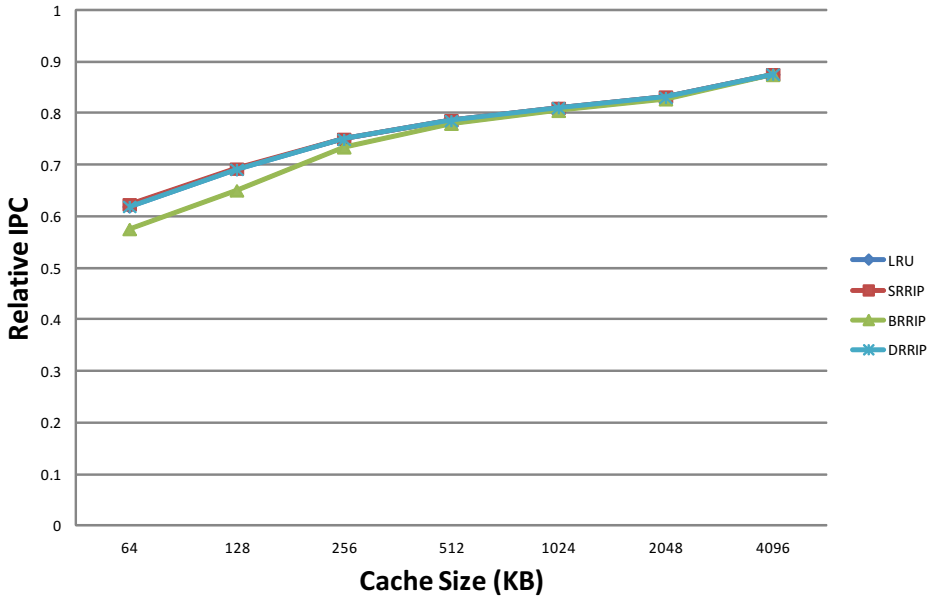

図 4.1.3: プリフェッチを行わない場合の実行性能の比較 (456.hmmer-ref.0)

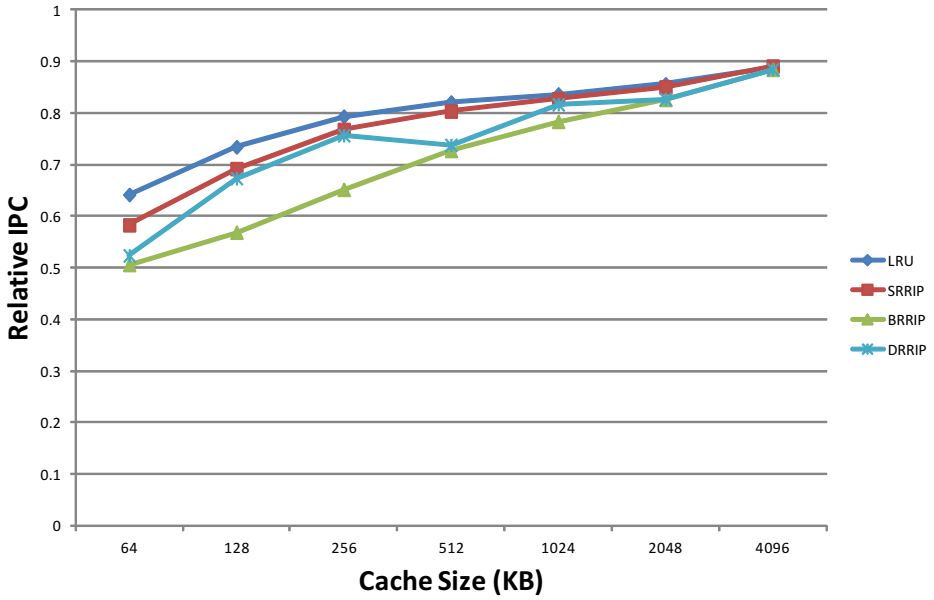

図 4.1.4: プリフェッチを行った場合の実行性能の比較 (456.hmmer-ref.0)

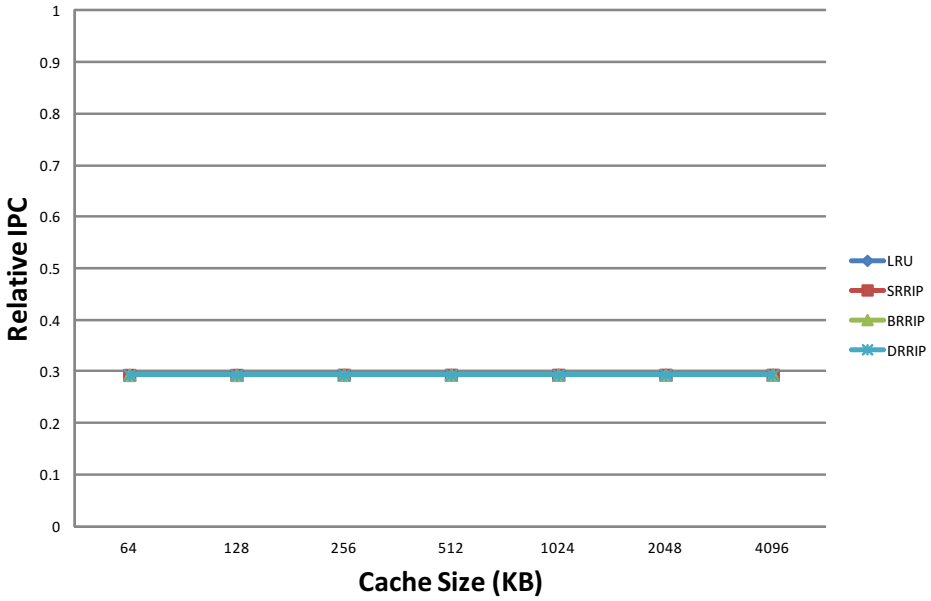

図 4.1.5: プリフェッチを行わない場合の実行性能の比較 (433.milc-ref.0)

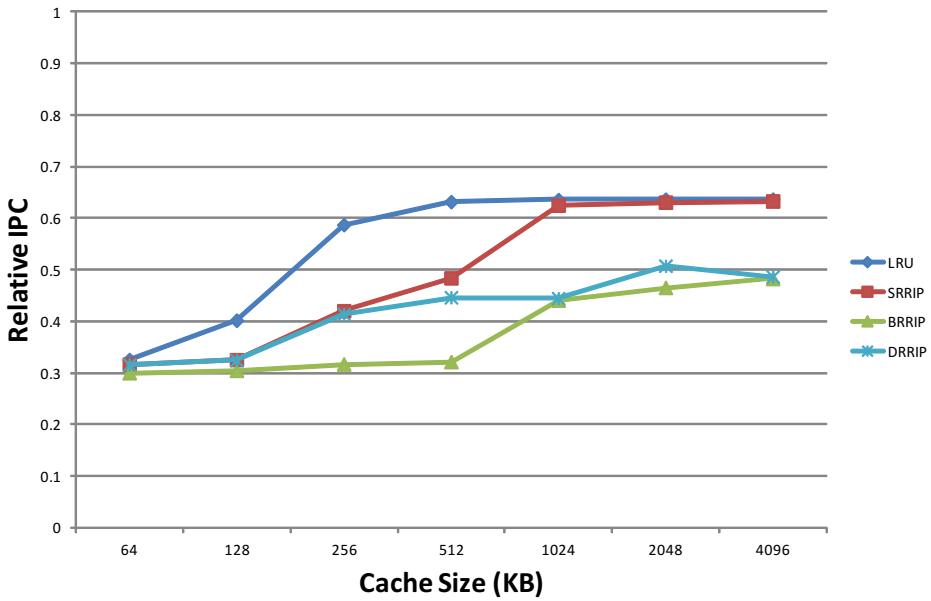

図 4.1.6: プリフェッチを行った場合の実行性能の比較 (433.milc-ref.0)

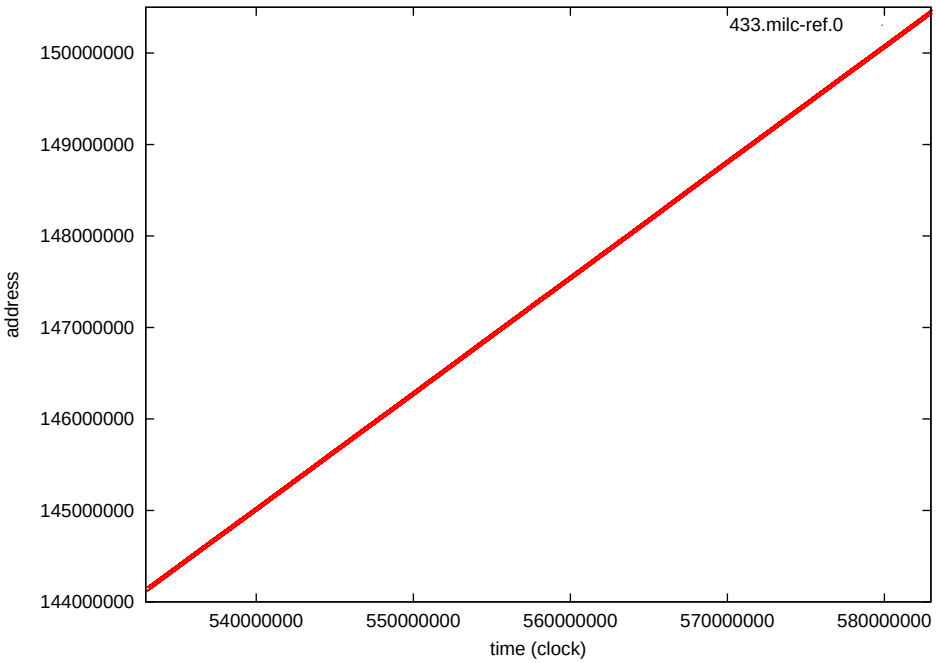

図 4.1.7: プリフェッチを行った LRU における L2 キャッシュへのアクセス (433.milc-ref.0)

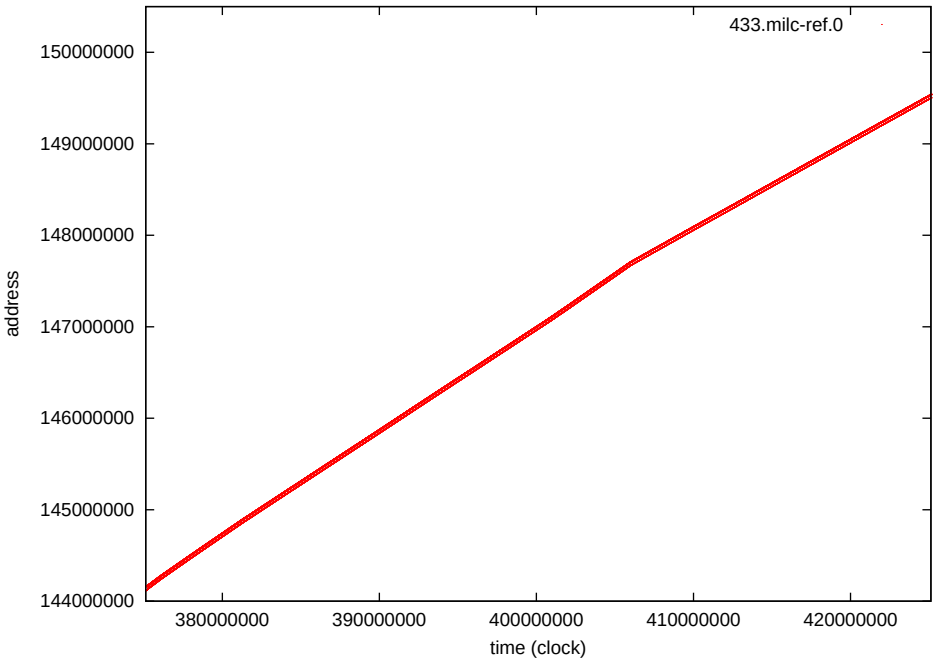

図 4.1.8: プリフェッチを行った SRRIP における L2 キャッシュへのアクセス (433.milc-ref.0)

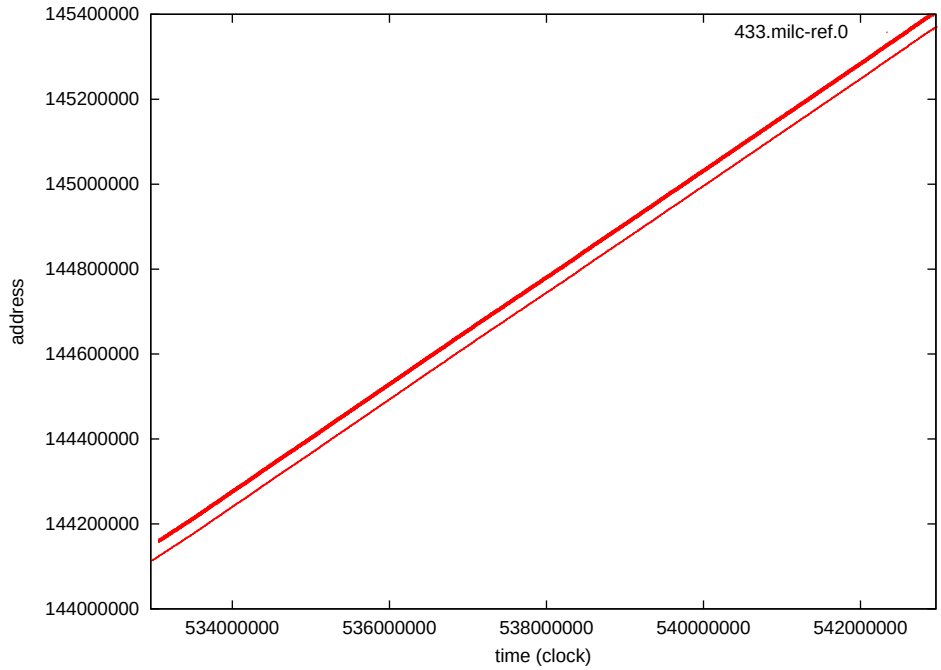

図 4.1.9: プリフェッチを行った LRU における L2 キャッシュへのアクセスの拡大図 (433.milc-ref.0)

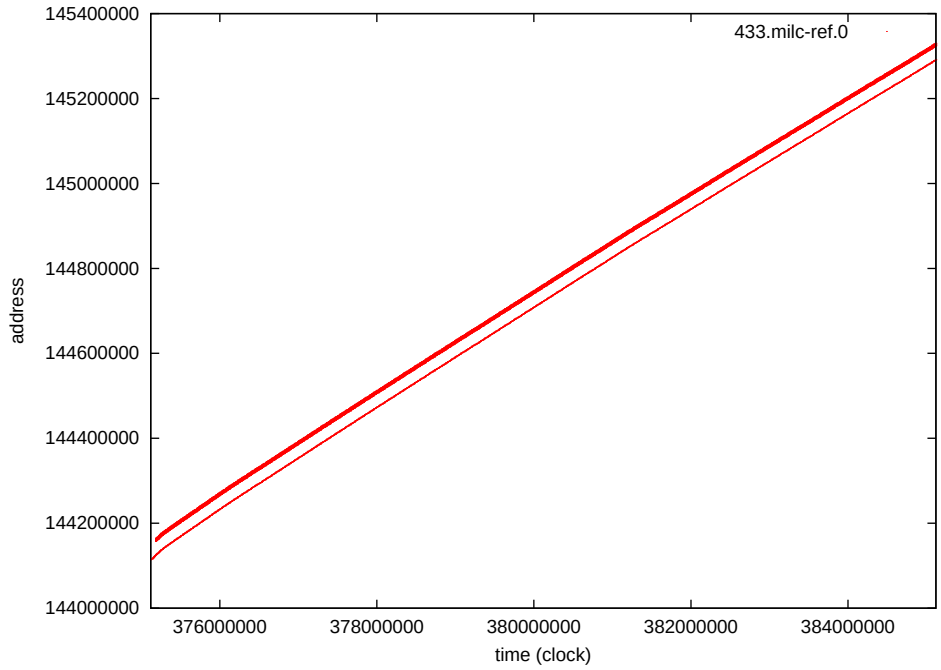

図 4.1.10: プリフェッチを行った SRRIP における L2 キャッシュへのアクセスの拡大図 (433.milc-ref.0)

した場合に，L2 キャッシュに入ってくるアクセスのアドレスと時間との関係を示したものである．lbm でも，近くのアドレスに対して，複数からアクセスされていることが確認できる．すなわち，同一のアドレスでデータが格納できず，データの入れ替えが頻繁に発生しているスラッシング状態になっている．これらにより SRRIP は LRU よりも IPC が低下したと言える．このように RRIP は高効率なアルゴリズムではあるが，プリフェッチを使用する場面ではキャッシュ性能に好ましい効果をもたらしておらず，キャッシュ性能の効率化のためには新たなアプローチが必要といえる．

4.2 プリフェッチを用いた新たなアプローチ

ライン置き換えとプリフェッチを組み合わせた新たなキャッシュアルゴリズムを提案する．

文献 [24] をはじめとして，プリフェッチを用いた既存手法では，プリフェッチでキャッシュに入れられたデータは再参照されず，ポリューションを生じさせていることに着目している．近い未来にアクセスされると予測されるデータに対してプリフェッチを行いアクセスレイテンシを隠蔽する一方で，プリフェッチによって入れられたデータはキャッシュから追い出されやすくするように制御を行なっている．

本研究では，キャッシュライン上に存在するデータの空間的局所性により再参照されるという価値を持っていることに着目した．プリフェッチによって得られるデータは，近い未来にアクセスされることが見込まれるデータであり，キャッシュを効率的に利用するためには必要なデータである．またキャッシュライン上に格納されているデータは局所性を持ち，近い未来にアクセスされるデータである．しかし，文献 [9] のアルゴリズムでは，特にキャッシュサイズが小さい場合において，ブロックが再参照される前にキャッシュラインから追い出されてしまう．そこで，近い未来に参照されるはずのデータがキャッシュから追い出されないようにするため，キャッシュライン上にあるデータが，プリフェッチによって得られたデータと一致するのであれば，データが近い将来再び使われるものとして，近い将来に参照されるはずのデータをキャッシュに残す制御を行う．

この制御を行うことで，キャッシュ性能を図 4.2.1 のように向上させる．図 4.2.2 に提案手法の概要を示す．具体的には，ハードウェアプリフェッチを行い，得られたデータのアドレスがキャッシュ内に格納されているデータのアドレスと一致した場合，一致したアドレスについて，追い出されないよう追い出し順序を変更する．この手法は LRU など他の置き換え手法にも適用することができるが，本論文では提案手法を 3.1.3 で述べた手法に対して適用した．プリフェッチによって得られたアドレスとキャッシュラインにあるアドレスが一致した場合，RRPV から-1 するようにすることで置き換え順序を変更した．

4.3 提案アルゴリズム

提案手法では次のように置き換えを行う．図 4.3.1 に RRIP に適用した提案手法の概要を，図 4.3.2 に SRRIP に適用した置き換えアルゴリズムを示す．

キャッシュヒット

1. 再参照によるヒットかプリフェッチによるヒットかを判別する
2. 再参照によるヒットの場合はヒットしたブロックの RRPV を 0 にする

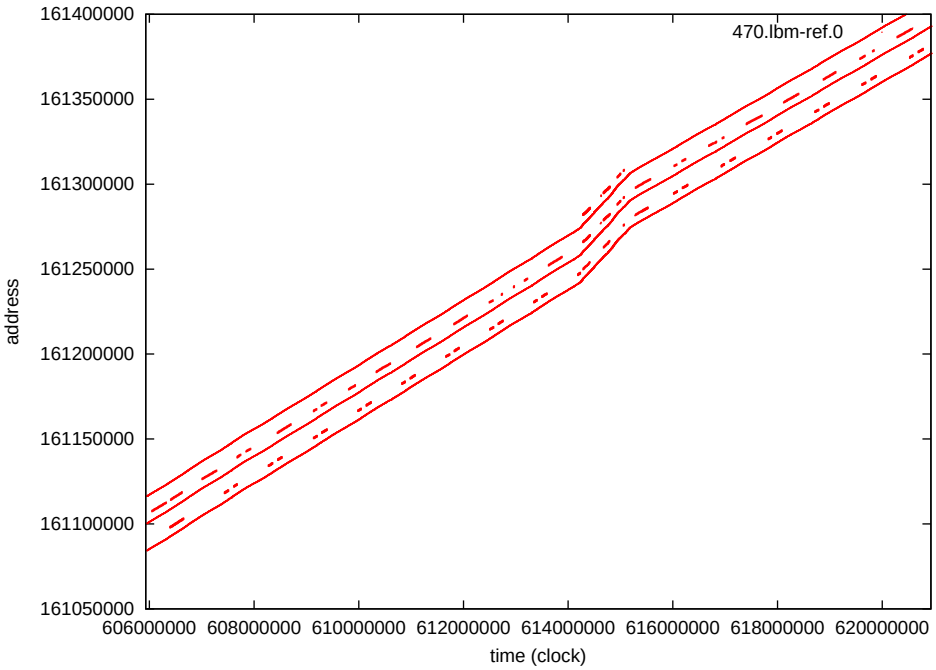

図 4.1.11: プリフェッチを行った LRU における L2 キャッシュへのアクセス (470.lbm-ref.0)

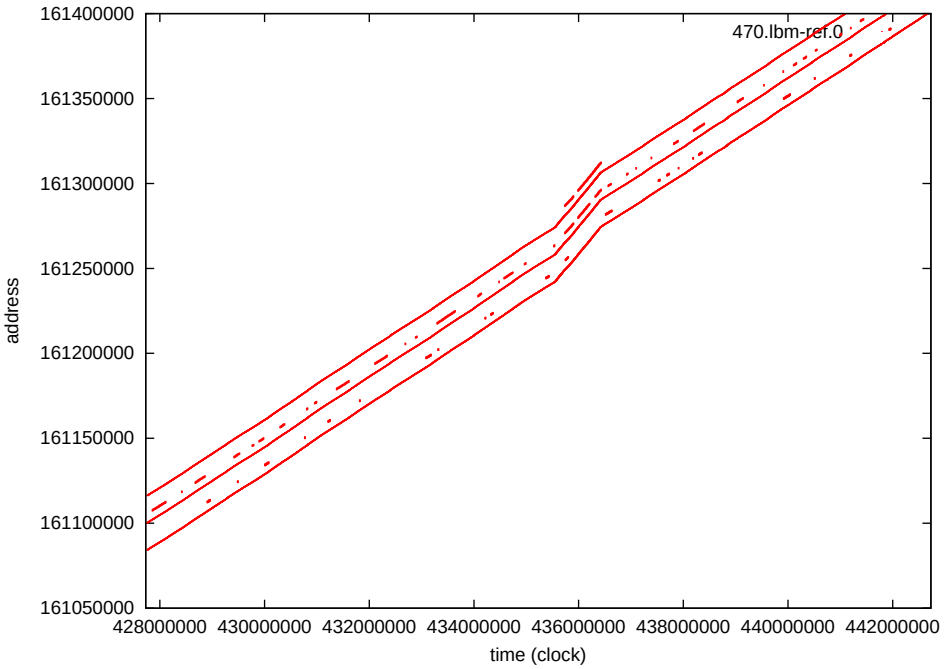

図 4.1.12: プリフェッチを行った SRRIP における L2 キャッシュへのアクセス (470.lbm-ref.0)

3. プリフェッチによるヒットの場合は，プリフェッチされたデータと一致したブロックの RRPV から 1 を引く．ただし，RRPV が既に 0 の場合は何もしない

キャッシュの置き換え

1. RRPV=3 のブロックを順に探索する
2. 空のブロックがあった場合は 4. を行う
3. RRPV=3 ブロックが見つかった場合はそのブロックを置き換える
4. いずれのブロックも見つからなかった場合は，全ての RRPV を +1 して，最初からやり直す
5. 新たに入ってきたデータの RRPV を 2 にする

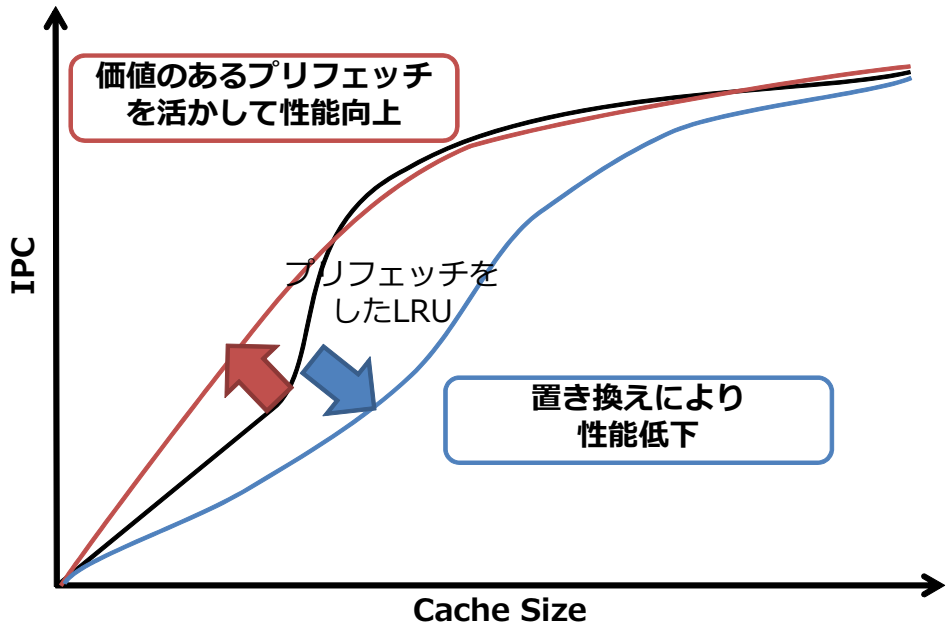

図 4.2.1: プリフェッチによりもたらされる効果

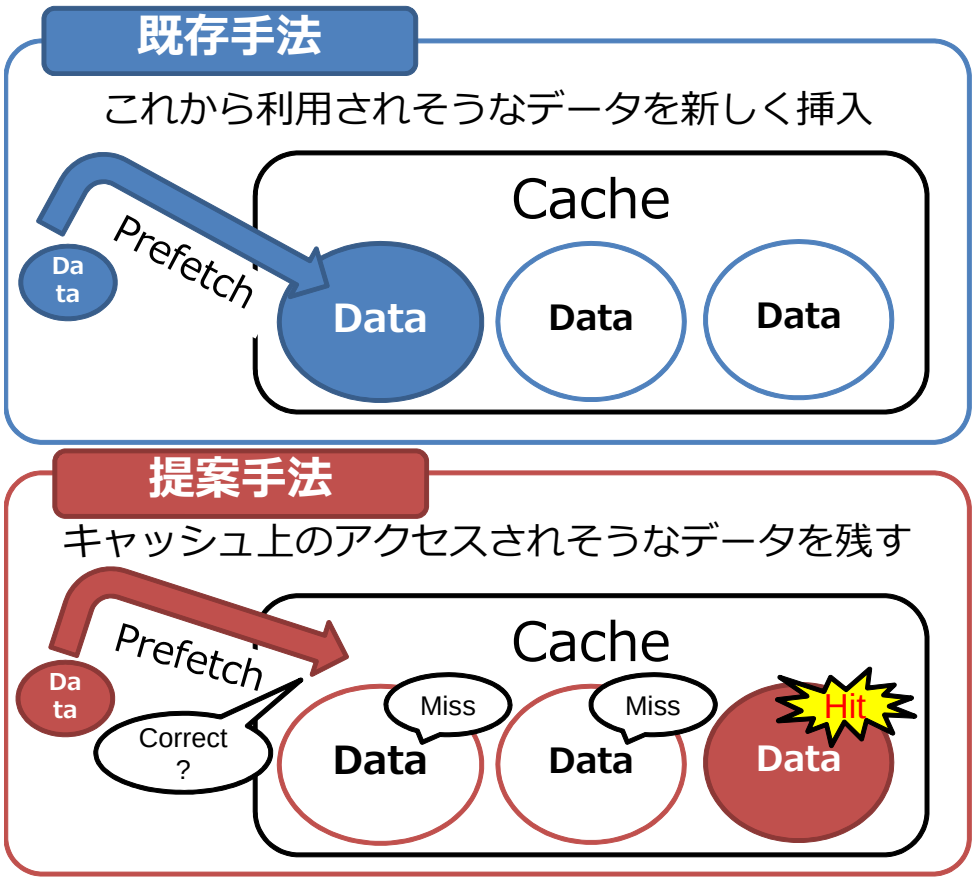

図 4.2.2: 提案手法の概要

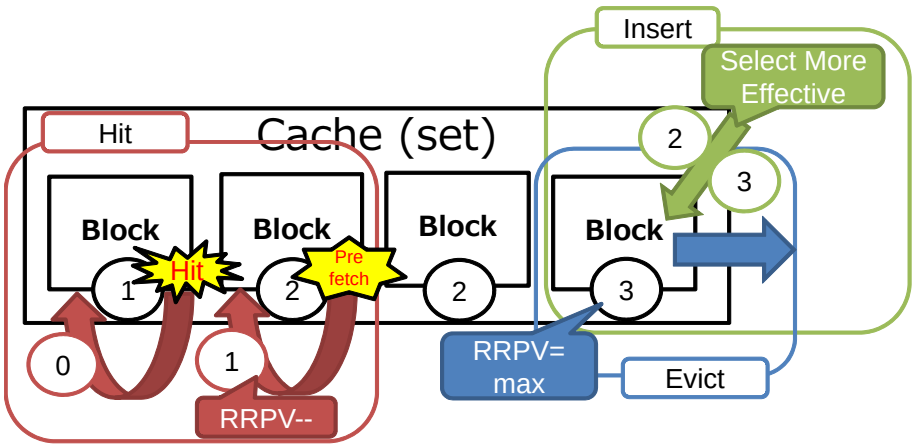

図 4.3.1: RRIP に適用した提案手法の概要

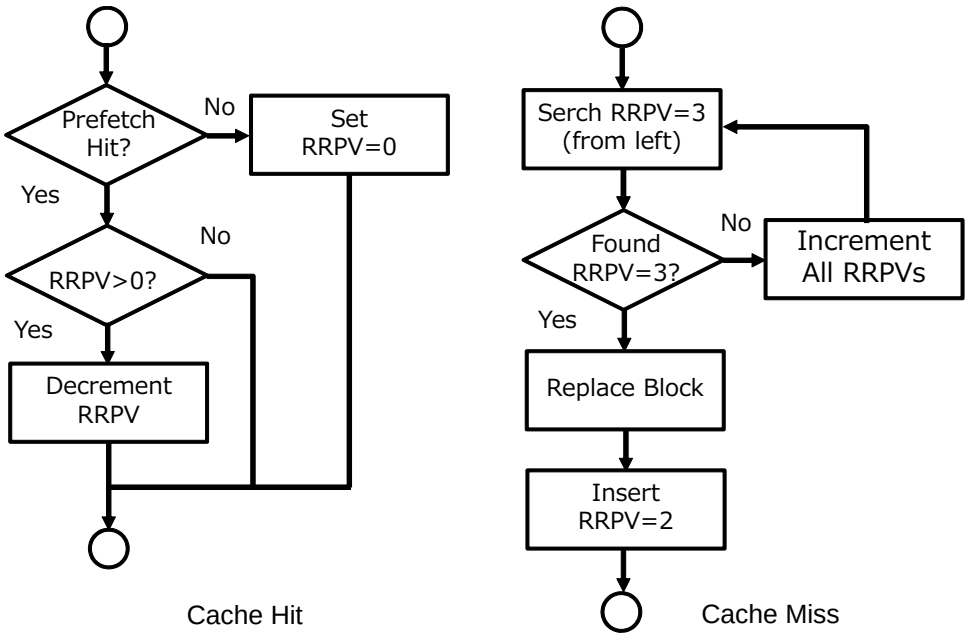

図 4.3.2: SRRIP に適用した置き換えアルゴリズム

第5章　実装

5.1　ハードウェア量

置き換えアルゴリズムで必要となるハードウェアについて検討する．ライン置き換えアルゴリズムでは各ラインについて，追い出しに用いる情報を格納するビットがタグアレイ内に必要となる．

RRIPでは，各ブロックに持たせたRRPVを用いて追い出す順序を決定している．RRPVは2bitの情報をway数だけ持つので，way数をwとすると，1セットあたりでは$2 \times w$ bit必要になる．DRRIPでは，RRPVに加えてSRRIPとBRRIPからミスが少ない方のアルゴリズムを選ぶため，Follower setsの識別に用いる5bitのカウンタと，Follower setsで使用するアルゴリズムを決定する10 bitのPESLが必要になる．

RRIPに適用した提案手法では，キャッシュヒットによりRRPVを更新する際に，プリフェッチによって得られたラインによるヒットかデマンド・アクセスによって得られたラインによるヒットかを識別できるようにする必要がある．しかし，この識別のためにタグアレイ内に新たな情報を持たせる必要はない．

5.2　既存手法とのハードウェア比較

LRUでは，セットごとにいつ参照されたかの情報を持たせて追い出し順序を決定している．順位の情報は，wayの数だけ必要であるため，1セットあたりで$w \log_2 w$ bit必要になる．既存手法であるPACManの場合，PACMan-Mでは，DRRIPと同様のハードウェアで実現できる．PACMan-H及びPACMan-HMでは，格納しているデータごとに，デマンド・アクセスによって入れられたものか，プリフェッチアクセスによって入れられたものかを記憶する必要がある．この情報には1 bitが必要であるので，1セットあたりでは$3 \times w$ bitだけ必要になる．またPACMan-DYNでは，3つのSDMを用いてFollower setsで使用するアルゴリズムを選択しているため，PACMan-HMで必要なレジスタに加えて，カウンタとPESLが3つずつ必要になる．

各アルゴリズムで必要になるレジスタの比較を表5.1に示す．提案手法はタグアレイに新たな容量を追加することなく実装できる．

表 5.1: 各手法で必要になるレジスタの比較

手法	セットごとに必要	全体で必要
LRU	$\log_2$ way $\times$ way [bit]	
SRRIP	$2 \times$ way [bit]	
$\text{SRRIP}_{\text{prefetch}}$	$2 \times$ way [bit]	
BRRIP	$2 \times$ way [bit]	
DRRIP	$2 \times$ way [bit]	15 [bit]
$\text{DRRIP}_{\text{prefetch}}$	$2 \times$ way [bit]	15 [bit]
PACMan-M	$2 \times$ way [bit]	15 [bit]
PACMan-H	$3 \times$ way [bit]	15 [bit]
PACMan-HM	$3 \times$ way [bit]	15 [bit]
PACMan-DYN	$3 \times$ way [bit]	15 [bit]

第6章　評価環境

6.1　ベースラインプロセッサ

プロセッサシミュレータ「鬼斬弐」rev.5321 [25, 26] に提案手法を実装した．また提案手法と比較する各手法を実装し，実行性能の評価を行った．プロセッサシミュレータのパラメタ設定を表 6.1 に示す．L1 キャッシュには LRU を用い，L2 キャッシュで 100%ヒットする場合のクロックあたりの実行命令数 (IPC) と評価に用いる置き換えアルゴリズムをそれぞれ適用した場合の IPC との割合を測定した．また，プリフェッチャにはシーケンシャルプリフェッチャを，L2 キャッシュに対して適用した．キャッシュミスが発生すると規則的なパターンを用いてデータをプリフェッチする．

6.2　比較手法

提案手法と既存手法との性能比較として，以下のアルゴリズムを実装し，L2 キャッシュに適用した上で評価を行った．

- プリフェッチを行った LRU
- プリフェッチを行った SRRIP
- プリフェッチを行った BRRIP
- プリフェッチを行った DRRIP
- PACMan-M
- PACMan-H
- PACMan-HM
- PACMan-DYN

表 6.1: ベースラインパラメタ

命令セット	
L1I キャッシュ	32KB, 4way, 3cycle, LRU
L1D キャッシュ	32KB, 4way, 3cycle, LRU
L2 キャッシュ	128KB, 8way,15cycle
メインメモリ	200cycle
プリフェッチャ	Stride Prefetcher, sequential degree 4

- SRRIP [9] に適用した提案手法 ($\text{SRRIP}_{\text{prefetch}}$)
- DRRIP に適用した提案手法 ($\text{DRRIP}_{\text{prefetch}}$)

なお，キャッシュライン挿入位置を選択パラメタ ϵ について，文献 [9, 24] に従い DRRIP, BRRIP では $\epsilon = 1/32$ を，PACMan では $\epsilon = 1/20$ をそれぞれ用いた．

6.3 ベンチマーク

ベンチマーク評価として，SPECCPU2006FP 及び INT のすべてのベンチマークを入力として用いた．ベンチマークのコンパイルには gcc version 4.2.2 を用い，-O3 オプションをつけて実行した．シミュレーションでは先頭 10G 回の実行命令をスキップし，その後の 100M 回の実行命令について測定を行った．

第7章　評価

7.1　実行性能の比較

図 7.1.1 は，シミュレータを用いて得られた各アルゴリズムの実行性能である．他のアルゴリズムと比べ，提案手法はキャッシュ性能が向上しており，$\text{SRRIP}_{\text{prefetch}}$ ではプリフェッチを行った SRRIP と比べ，29 本のプログラムのうち 18 本で性能が向上し．最大で 42%，またベンチマーク全体での幾何平均で 2.2% 高くなった．$\text{DRRIP}_{\text{prefetch}}$ についてもプリフェッチを行った DRRIP と比べ，最大で 30%，幾何平均で 4.6%の性能向上が得られた．また，プリフェッチを行った LRU に対しては，$\text{SRRIP}_{\text{prefetch}}$ は 1.3%高くなり，$\text{DRRIP}_{\text{prefetch}}$ は逆に 0.2%低くなった．

図 7.1.2，図 7.1.3 は，ベンチマークとして 433.milc と 470.lbm を選び，$\text{SRRIP}_{\text{prefetch}}$ を用いた際に，L2 キャッシュに入るアクセスと時間との関係を示したものである．milc について見ると，図 4.1.8 からアドレスとクロックとの関係の変化があり，clock が 504000000 以降では SRRIP にプリフェッチを行ったものと同じく傾斜が緩やかだが，それ以前では傾斜が急になり，キャッシュの処理効率が改善したことが見て取れる． また lbm について見ると，図 4.1.12 と比べ，断続的にあった点のようなアクセスの数が減少していることが確認できる．これはスラッシングによる悪影響が改善したと言える．

7.2　キャッシュサイズによる変化

L2 のキャッシュサイズを変えて評価を行った．図 7.2.1 は SPECCPU2006 の全てのプログラムについてプリフェッチを行い，L2 キャッシュのサイズを 64KB から 4MB まで変化させ実行性能を比較したものである．また，図 7.2.2 は 433.milc を選び，同様に比較したものである． 幾何平均で見ると，提案手法はいずれも特にキャッシュサイズが小さい場合においてプリフェッチを行った SRRIP や DRRIP に対して，1%から 2%程度高くなり，プリフェッチを行った LRU に対しても高くなった．一方で，キャッシュサイズが大きくなると性能が逆に下がっており，特に L2 キャッシュが 256KB の時での $\text{DRRIP}_{\text{prefetch}}$ では 1.5%下がっている．433.milc で見ると，提案手法によりプリフェッチを行った SRRIP や DRRIP に対して，30%程度高くなっている．

7.3　プリフェッチ積極度による変化

プリフェッチャの積極度を変えて評価を行った．図 7.3.1 は SPECCPU2006 の全てのプログラムについて，プリフェッチの sequential degree を 16 に変化させた際の実行性能を比較したものである．幾何平均を見ると，$\text{SRRIP}_{\text{prefetch}}$ はプリフェッチを行った SRRIP に対して 2.1%，$\text{DRRIP}_{\text{prefetch}}$ ではプリフェッチを行った DRRIP に対して 5.1%高くなった．また，プリフェッチを行った LRU に対しては，$\text{SRRIP}_{\text{prefetch}}$ は 0.5%高くなり，$\text{DRRIP}_{\text{prefetch}}$ は 0.9%低くなった．

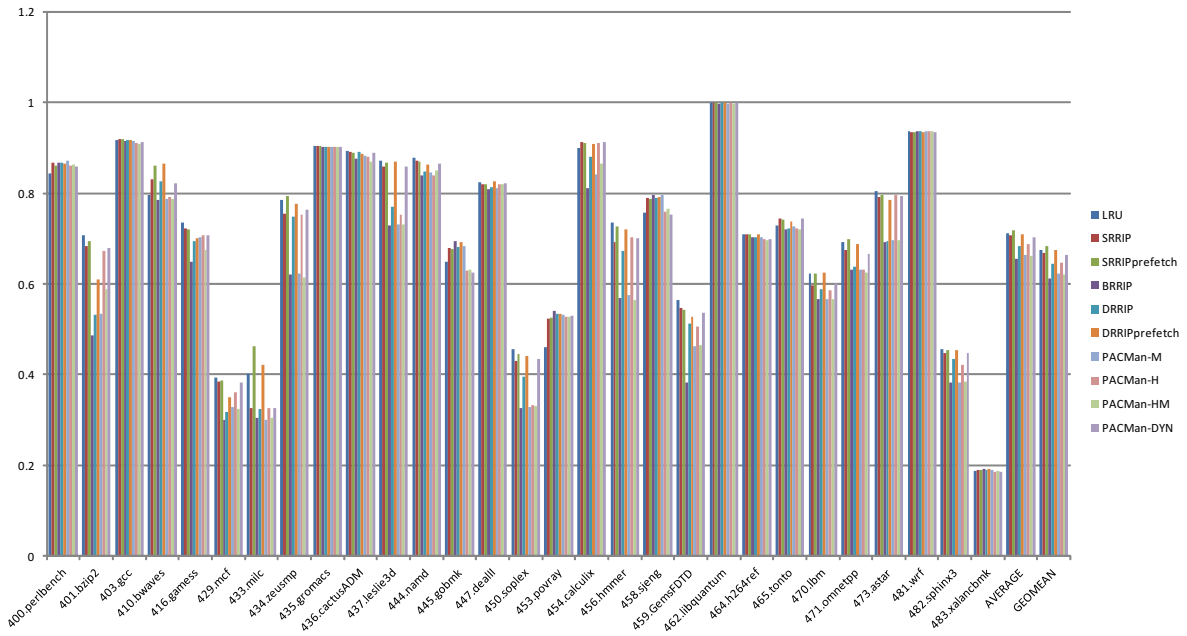

図 7.1.1: 提案手法との性能比較

このように，プリフェッチを用いてキャッシュに格納されているアクセスされるデータを追い出されにくくすることで，プリフェッチの恩恵が受けられなかった既存手法に対して，キャッシュ性能を向上させることができた．

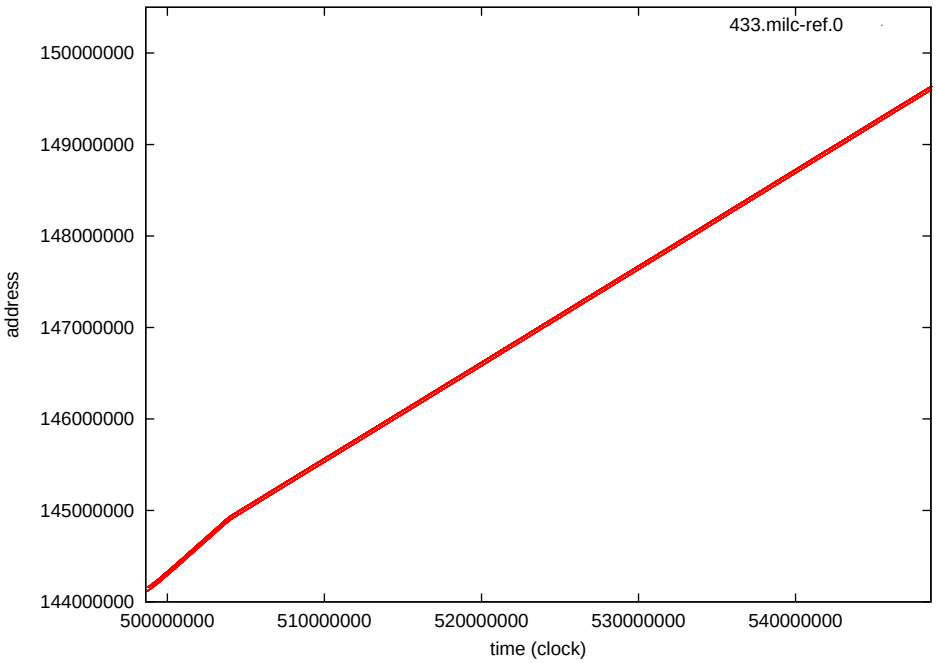

図 7.1.2: SRRIP を適用した提案手法における L2 キャッシュへのアクセス (433.milc-ref.0)

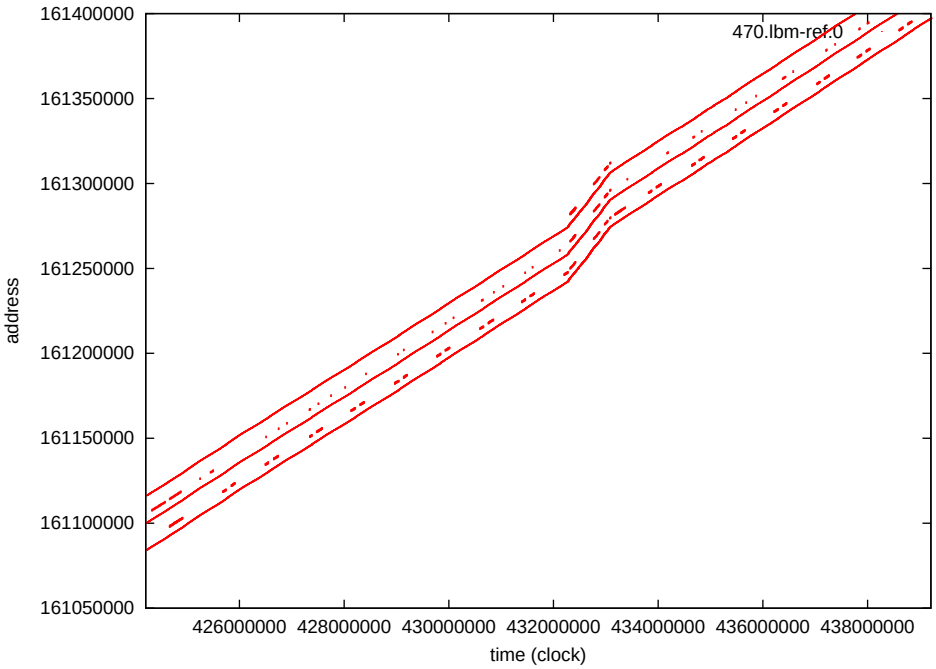

図 7.1.3: SRRIP を適用した提案手法における L2 キャッシュへのアクセス (470.lbm-ref.0)

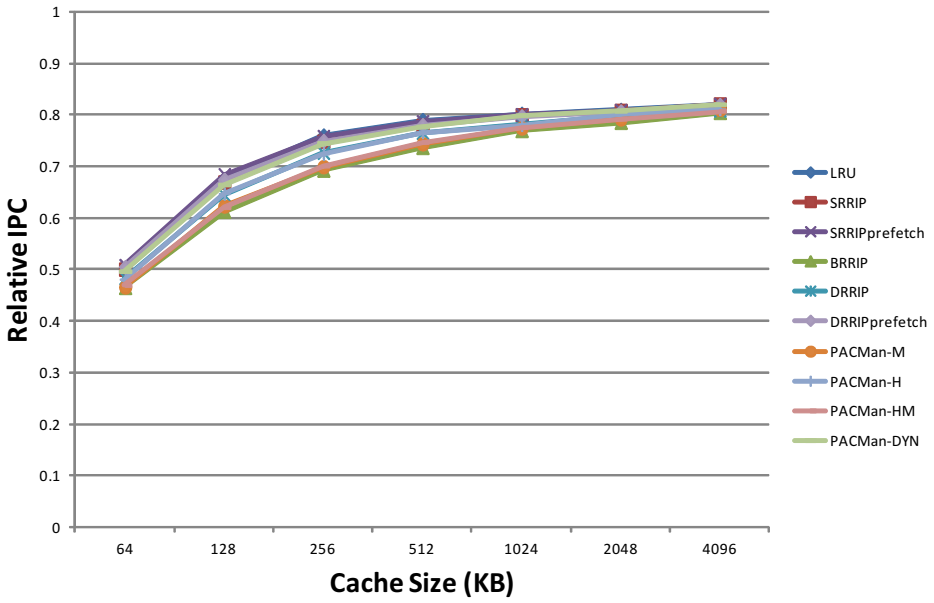

図 7.2.1: L2 キャッシュのサイズを変化させた場合の実行性能の比較

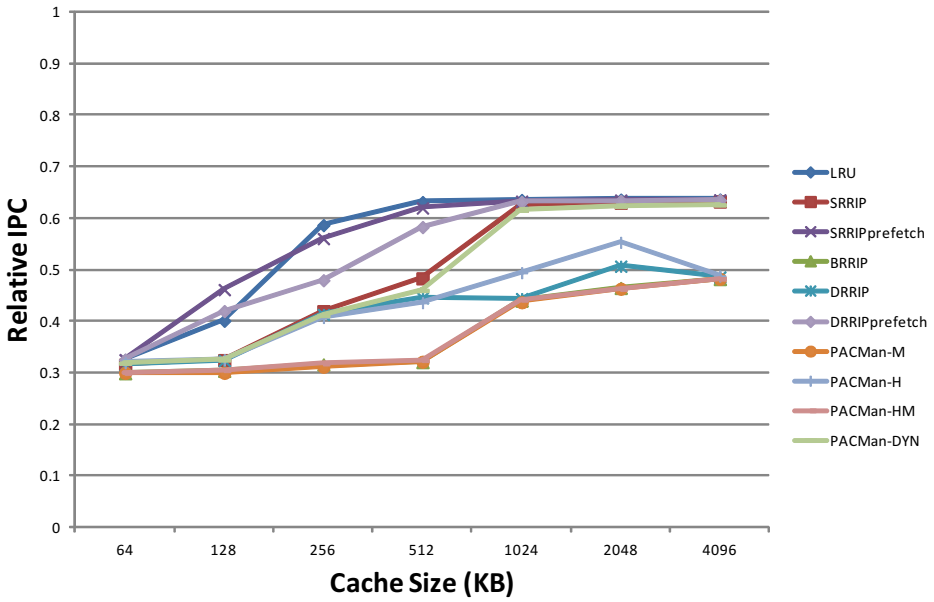

図 7.2.2: L2 キャッシュのサイズを変化させた場合の実行性能の比較 (433.milc-ref.0)

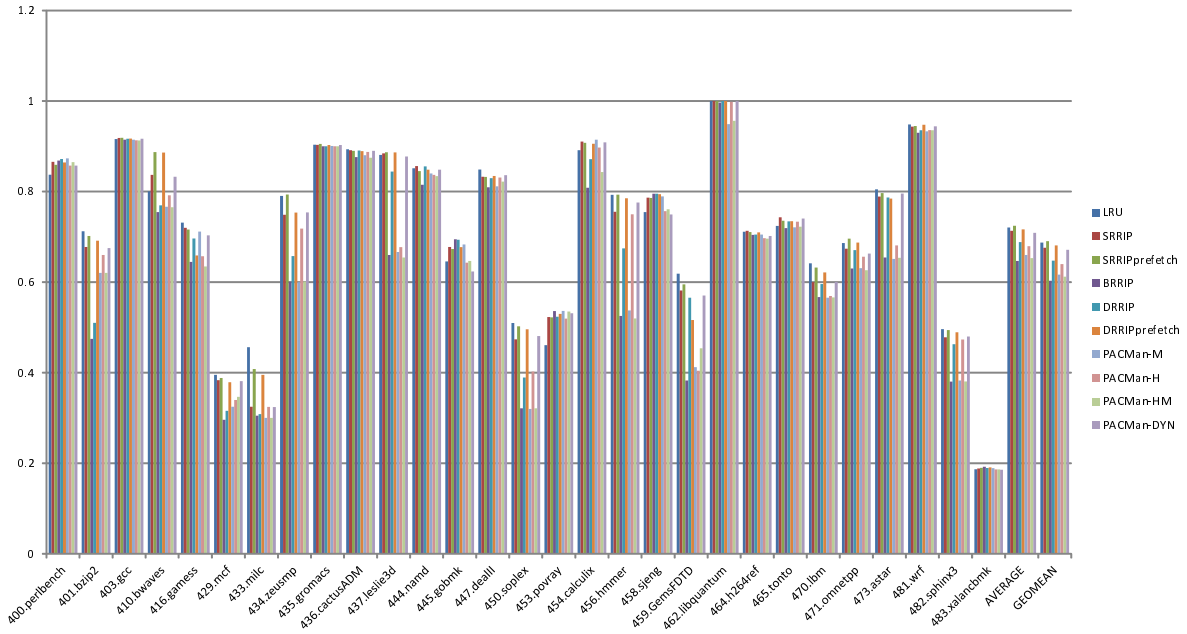

図 7.3.1: プリフェッチ積極度を変化させた場合の実行性能の比較

第8章　結論

計算機の性能向上には，キャッシュメモリを用いて，局所性のあるデータをうまく管理することが重要である．キャッシュをうまく管理する方法として，データの置き換えやプリフェッチが広く使われている．しかし，現在提案されている手法では，特にキャッシュサイズが小さい場合に，プリフェッチを用いても再参照される前にキャッシュから追い出されてしまい，性能が向上できていない．

本研究では，プリフェッチによりキャッシュラインに入ってくるデータに着目した．プリフェッチにより得られたデータは，近い未来に参照され，キャッシュ性能向上のために役立つものである．そこで．プリフェッチによって得られたデータがキャッシュ内に格納されていた場合，キャッシュから追い出されにくくなるように制御を行う新たなアプローチを提案した．シミュレータを用いた評価では，既存のキャッシュアルゴリズムと比べ最大で 42%，幾何平均でも 2.2% の性能向上が得られた．

今後の課題として，プリフェッチ量の動的な調節やマルチコアへの対応が挙げられる．

謝辞

本研究にあたり電気通信大学情報システム学研究科情報ネットワークシステム学専攻入江英嗣准教授に終始，ご指導を頂いた． ここに深謝の意を表する．また，同専攻吉永・入江研究室の各位，特に吉永努教授，吉見真聡助教，放地宏佳君，藤原大輔君には研究遂行にあたり日頃より有益なご討論ご助言を戴いた．ここに感謝の意を表する．

参考文献

[1] ヘネシージョン・L. コンピュータアーキテクチャ定量的アプローチ第 4 版:. IT architects' archive. 翔泳社, 2008.

[2] A.J. Smith. Cache memories. *ACM Computing Surveys (CSUR)*, Vol. 14, No. 3, pp. 473–530, 1982.

[3] E.J. O'neil, P.E. O'neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *ACM SIGMOD Record*, Vol. 22, pp. 297–306. ACM, 1993.

[4] D. Lee, J. Choi, J.H. Kim, S.H. Noh, S.L. Min, Y. Cho, C.S. Kim, et al. LRFU (least recently/frequently used) replacement policy: A spectrum of block replacement policies. *In IEEE Transactions on Computers. Citeseer*, 1996.

[5] T. Johnson and D. Shasha. 2Q: A low overhead high performance bu er management replacement algorithm. 2004.

[6] N. Megiddo and D.S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pp. 115–130, 2003.

[7] M.K. Qureshi, A. Jaleel, Y.N. Patt, S.C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ACM SIGARCH Computer Architecture News*, Vol. 35, pp. 381–391. ACM, 2007.

[8] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely Jr, and J. Emer. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 208–219. ACM, 2008.

[9] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pp. 60–71, New York, NY, USA, 2010. ACM.

[10] P. Michaud. The 3p and 4p cache replacement policies. In *JWAC 2010-1st JILP Worshop on Computer Architecture Competitions: cache replacement Championship*, 2010.

[11] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. SHiP: signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pp. 430–441, New York, NY, USA, 2011. ACM.

[12] 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. Technical report, Technical Report SAFARI 2012-002, CMU, 2012.

[13] A.C. Lai, C. Fide, and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, pp. 144–154. IEEE, 2001.

[14] M. Kharbutli and Y. Solihin. Counter-based cache replacement and bypassing algorithms. *Computers, IEEE Transactions on*, Vol. 57, No. 4, pp. 433–447, 2008.

[15] Z. Hu, S. Kaxiras, and M. Martonosi. Timekeeping in the memory system: predicting and optimizing memory behavior. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pp. 209–220. IEEE, 2002.

[16] J. Abella, A. González, X. Vera, and M.F.P. O'Boyle. IATAC: a smart predictor to turn-off l 2 cache lines. *ACM Transactions on Architecture and Code Optimization*, Vol. 2, No. 1, pp. 55–77, 2005.

[17] Haiming Liu, Michael Ferdman, Jaehyuk Huh, and Doug 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*, MICRO 41, pp. 222–233, Washington, DC, USA, 2008. IEEE Computer Society.

[18] Z. Wang, D. Burger, K.S. McKinley, S.K. Reinhardt, and C.C. Weems. Guided region prefetching: a cooperative hardware/software approach. In *ACM SIGARCH Computer Architecture News*, Vol. 31, pp. 388–398. ACM, 2003.

[19] 堀部悠平, 張鵬, 小笠原嘉泰, 三輪忍, 中條拓伯. メモリ・アクセス・パターンを利用した高精度ハードウェア・プリフェッチ手法 (アクセラレーション/メモリシステム,「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ (hokke-2009)). 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, Vol. 2009, No. 14, pp. 91–96, 2009.

[20] E. Ebrahimi, O. Mutlu, and Y.N. Patt. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 7–17. IEEE, 2009.

[21] 入江英嗣, 本城剛毅, 平木敬. 動的推定によるプリフェッチ量最適化. 情報処理学会論文誌. コンピューティングシステム, Vol. 3, No. 3, pp. 56–66, sep 2010.

[22] 石井康雄, 稲葉真理, 平木敬. 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. 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, Vol. 2010, No. 13, pp. 1–8, 2010.

[23] K. Zhang, Z. Wang, Y. Chen, H. Zhu, and X.H. Sun. Pac-plru: A cache replacement policy to salvage discarded predictions from hardware prefetchers. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 265–274. IEEE, 2011.

[24] Carole-Jean Wu, Aamer Jaleel, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. Pacman: prefetch-aware cache management for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pp. 442–453, New York, NY, USA, 2011. ACM.

[25] 渡辺憲一, 一林宏憲, 五島正裕, 坂井修一. プロセッサ・シミュレータ「鬼斬」の設計. 先進的計算基盤システムシンポジウム SACSIS, pp. 194–195, 2007.

[26] 塩谷亮太, 五島正裕, 坂井修一. プロセッサ・シミュレータ「鬼斬弐」の設計と実装. 先進的計算基盤システムシンポジウム SACSIS, 2009.

発表文献

[1] 入江 英嗣, 放地 宏佳, 小木 真人, 樫原 裕大, 芝 星帆, 眞島 一貴:「AirTarget: 光学シースルー方式 HMD とマーカレス画像認識による高可搬性実世界志向インターフェース」, マルチメディア, 分散, 協調とモバイル (DICOMO2012) シンポジウム, pp.1295-1304, Jul. 2012.

[2] Hidetsugu IRIE, Daisuke FUJIWARA, Kazuki MAJIMA, and Tsutomu YOSHINAGA: STRAIGHT: Realizing a Lightweight Large Instruction Window by using Eventually Consistent Distributed Registers, Int. Workshop on Challenges on Massively Parallel Processors, pp.336-34. Dec. 2012.

[3] 稲場 朋大，放地 宏佳，藤原 大輔，眞島 一貴，吉見 真聡，入江 英嗣，吉永 努:「配線アクティビティを考慮した 3 次元積層プロセッサ向けフロアプランナーのための熱評価手法」, 情報処理学会研究報告，2012-ARC-203（2 月発表予定）

[4] 入江 英嗣，放地 宏佳，稲場 朋大，眞島 一貴，藤原 大輔，吉見 真聡，吉永 努:「配線アクティビティを考慮した 3 次元積層プロセッサ向けフロアプランナー」，情報処理学会論文誌 Vol.53 No.2 , Feb. 2012.（投稿中）

[5] 入江 英嗣，藤原 大輔，眞島 一貴，稲場 朋大，吉見 真聡，吉永 努:「もし ILP プロセッサのレジスタファイルが分散キーバリューストアになったら」，先進的計算基盤システムシンポジウム SACSIS2013.（投稿中）