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	野村隼人	学籍番号	1452018
論 文 題 目	長期再参照のためのキャッシュライン追い出し不可属性の提案		

要　旨

プロセッサアーキテクチャにおいて，メモリサブシステム，キャッシュシステムの性能は依然として主要な課題の一つである．近年では半導体微細化に伴いキャッシュメモリ，特にラストレベルキャッシュ（Last Level Cache, LLC）の大容量化が進み，これを上手く利用するためにプリフェッチに協調するスマートなキャッシュ置き換えアルゴリズムが提案されている．

しかしながら，これらの賢いキャッシュアルゴリズムをもってしても救えていないキャッシュミスが存在する．本研究の調査で，それは一度アクセスされたあるアドレスに対応するラインがキャッシュに載った後，実行命令数が 1M 命令以上経過した後に再度アクセスされた場合に生じるキャッシュミスであることが明らかになった．本研究ではこれを長期再参照ミスと命名した．これらの調査から, 特に LLC でのキャッシュミスのボリュームゾーンは長期再参照ミスであることがわかった．

この調査を踏まえて，本研究では，ある基準によって定めたキャッシュラインを長期間追い出さないことを特長としたキャッシュアルゴリズムの基本戦略を提案する．実行履歴に基づき置き換えを起こすアルゴリズムによって救えないキャッシュミスであるならば，いっそヒストリに頼った直近の再参照に期待するのはやめて，置き換えを起こさない手法として試行した．我々はこれに Stubborn 戦略と名付けた．“Stubborn”は“頑固”を意味する． Stubborn 戦略は，キャッシュ中に追い出しを起こさない領域をつくることで実現させている．

本提案手法を，LRU をベースに実装し，シミュレーションによる評価を行った．このようなシンプルな戦略にも関わらず，結果として最大で 23.9%の IPC 向上，幾何平均でも 2MB 構成，Prefetch aware な LRU ベースでのランダム採択を行う Stubborn キャッシュで最大の 2.40%向上が得られた．

平成 27 年度修士論文

長期再参照のための
キャッシュライン追い出し不可属性の提案

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：　1452018
氏名：　野村 隼人
主任指導教員：吉永 努　教授
指導教員：　長岡 浩司 教授
指導教員：　笠井 裕之 准教授
提出年月日：　平成 28 年 1 月 28 日

(表紙裏)

目次

図目次

表目次

第1章 はじめに

1.1. 研究背景

プロセッサアーキテクチャにおいて，メモリサブシステム，キャッシュシステムの性能は主要な課題の一つである[1]．近年では低次キャッシュ，特にラストレベルキャッシュ（Last Level Cache, LLC）の大容量化が進み，従来のシンプルな局所性利用アルゴリズムを発展させたスマートなキャッシュ制御アルゴリズムの研究が数多く進められている．これまでのプロセッサ上に搭載するキャッシュメモリを対象としたキャッシュアルゴリズムの研究では，2 つの重要なテクニックとして，置き換えアルゴリズムとプリフェッチが存在する．置き換えアルゴリズムは，各キャッシュラインの重要度を判断しキャッシュに残すべきデータをアクセス履歴から判断し，時間局所性のあるプログラムのキャッシュアクセスを支援する．具体的な実装例としては LRU [16,25]，RRIP [3] が挙げられる．また，プリフェッチ [7,9] は，過去のアクセスパターンから次にアクセスされるデータのアドレスの推定を行い，大容量化した LLC を活用してあらかじめキャッシュに格納することで初期参照ミスを隠蔽し，性能の向上を得ている．このように，プリフェッチが LLC のキャッシュ容量を有効活用することで性能向上をもたらし，スマートな追い出しアルゴリズムがプリフェッチによるデッドブロックから保護すべき価値のあるデータをキャッシュに残す仕組みが現在のキャッシュシステムに高い性能を生み出している．

1.2. 研究目的

本研究では，従来のキャッシュアルゴリズムでは救うことができなかったキャッシュミスについて調査した．プリフェッチと，スマートな置き換えアルゴリズムはこれまでキャッシュミスを削減し，性能向上を推し進めてきた．しかしながら，それでもなお救いきれないキャッシュミスが存在する．我々の調査において，それは**長期再参照ミス**であると断定した．それは，長期間で繰り返し再参照されるアドレスで発生するキャッシュミスであり，そのようなキャッシュミスは，1M 命令から 10M 命令程度の間隔を空けて生じる．本研究の目的は，長期再参照によるキャッシュミスを生じるデータを救うことで，これまでのキャッシュアルゴリズムでは防ぐことができなかったキャッシュミスを削減することができるか，それによって得られる性能向上はどれ

ほどのものか，を検証することである．また，その過程として長期再参照ミスがどのような特性を持つのかについて調査する．

本研究では，長期再参照を生じるデータを救済する仕組みとして，**Stubborn 戦略**と，それに基づく **Stubborn キャッシュ**を提案する．この Stubborn 戦略とは **1G 命令程度の長期間，あるキャッシュライン群を追い出さずにずっと持ち続ける**ことで長期再参照を生じるデータを持つキャッシュラインのミスを救う手法である．Stubborn 戦略を，ハードウェア実現可能な追い出し不可属性付与対象キャッシュライン選択アルゴリズムと共に実装した場合における正確な効果を明らかにする．

1.3. 本論文の構成

従来のキャッシュアルゴリズムで救えないキャッシュミスについての調査と，そこから導き出された長期再参照ミスの実態の調査．それを踏まえて Stubborn 戦略の提案を行い，Stubborn 戦略の持つポテンシャルを測る．そのために，Stubborn 戦略が最も理想的な条件の下で最適に働いた場合のミス削減効果を調べる．

次に，Stubborn 戦略のための実現可能なキャッシュライン選択アルゴリズムの開発を行う．これよって達成される性能向上率と，キャッシュミスの削減率を評価するため，より正確なシミュレーションを行うシミュレーション環境による評価を行う．キャッシュされているデータの遷移の観察を行い，これまでのキャッシュアルゴリズムでは対処できなかった長期再参照でキャッシュミスが救えているかどうかを確認する．最終的には命令実行効率の指標である IPC (Instructions Per Cycle)での性能向上率の評価を行う．

まず第 1 章では，本研究の技術的な背景について議論し，さらに目的や課題について論じる．続いて第 2 章にて，本研究の対象となるキャッシュメモリがどのような構成であるか述べ, 第 3 章で従来のキャッシュアルゴリズムがどのようなものであるか, どういった特性をもつか，について議論する．さらに，第 4 章では従来のキャッシュアルゴリズムでは救うことができないキャッシュアクセスパターンがどのようなものであるかを調査した結果として，長距離再参照ミスについて述べる．この調査を踏まえ，第 5 章では長距離再参照ミスを救うためのキャッシュアルゴリズムの戦略である Stubborn 戦略を提案する．第 6 章では理想的な条件で評価を行い，Stubborn 戦略のもつポテンシャルがいかほどであるかを評価する．第 7 章では，Stubborn 戦略を実計算機として実現可能な形態にするためにどのような実装とするかを述べる．第 8 章で定義した評価環境を基本として，第 9 章ではその構成に基づく Stubborn 戦略を実装したキャッシュシステムの評価結果について示し，考察する．第 10 章では，今後の展望について議論し，第 11 章では本研究の成果をまとめる．

第2章 キャッシュメモリ

本章では，本論文でキャッシュメモリについて議論するにあたって，前提知識となるキャシュメモリの概要を解説する．キャッシュメモリのコンピュータアーキテクチャ中における役割，キャッシュメモリの構成，ハードウェア的な構造について順に述べる．また，本論文でキャッシュ構成と評価の議論に使用する，階層化キャッシュ，Last level cache (LLC)，ライン，セット，ウェイ，インデックス，オフセットビット，キャッシュミス，初期参照，再参照，MPKI，IPC のそれぞれの語についても解説する．

2.1. キャッシュメモリの概要

キャッシュメモリは，メインメモリよりも高速で小さな容量の記憶領域であり，CPU コア（主に演算器，命令デコーダ）とメインメモリの間の転送遅延を隠蔽する働きをする．近年の多くのプロセッサではキャッシュメモリは階層化構造となっており，CPU コアに近い順から Level 1 キャッシュ，Level 2 キャッシュ， Level 3 キャッシュと呼ばれる．特に，最もメインメモリに近い層であるキャッシュメモリは Last level cache (LLC)と呼ばれ，現在でのハイエンドプロセッサでは 1MB から 20MB 程度の MB オーダーでの容量をもつ．キャッシュメモリは，CPU コアに近いほど読み書きレイテンシが小さく，逆にメインメモリに近い方へのアクセスほどレイテンシが大きくなる．

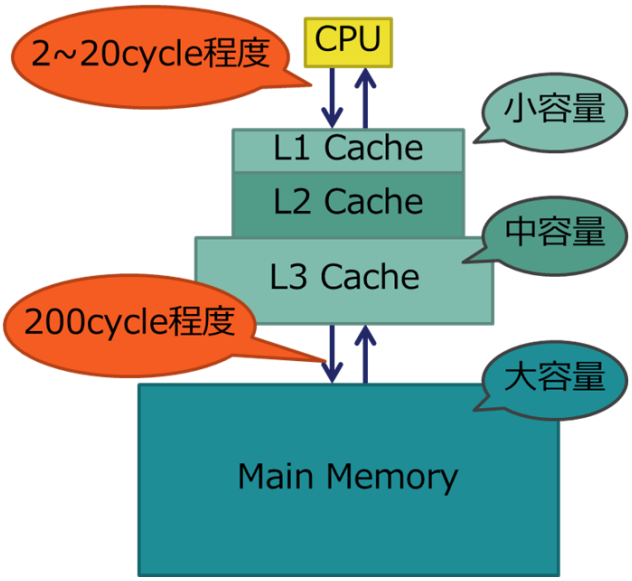

図 2.1　階層化されたメモリシステムと階層ごとの容量，レイテンシ

2.2. キャッシュメモリの構成

次に，キャッシュ上でのデータの配置の実体を説明する．現在のキャッシュメモリを構成するテーブルは，図 2.2 のようなセットアソシアティブな配置方式が主流となっている．データが格納される単位はラインであり，セットアソシアティブな構成ではラインが収まる行をセット，1 セットあたりの列の数をウェイと呼ぶ.

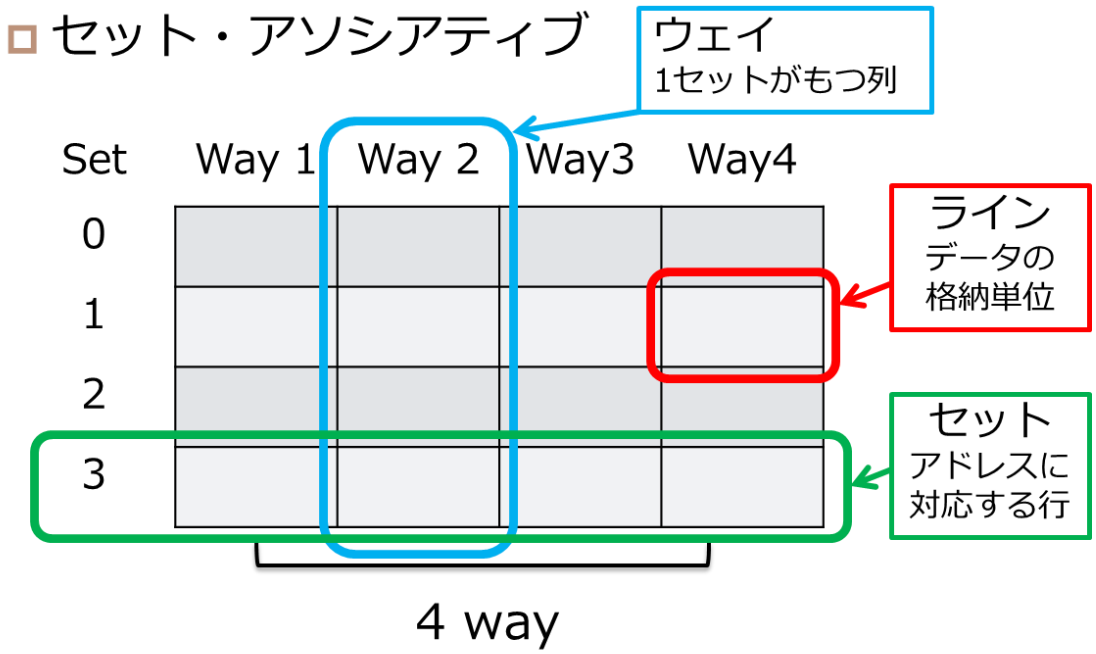

図 2.2　セットアソシアティブなキャッシュテーブルの構造

次に, プログラムのアドレスとキャッシュへの格納時の位置対応について説明する. 図 2.3 ではあるアドレスへのアクセスにより，キャッシュへ該当データがあるかどうかを問い合わせるタイミングを想定する． アクセスがあったアドレスを，上位からタグ，インデックス，オフセットに分解し，インデックスをキャッシュテーブルのセット数に収まるようなハッシュにかけて，アドレスに対応するキャッシュラインが格納されうるセットを選定する. インデックスによりアクセスするセットが決まる. 次に，そのセット内でタグが一致するラインを探す．タグが一致するものがあればキャッシュヒットで，該当するラインからデータを取り出す．なければキャッシュミスで，そのキャッシュよりさらに下の階層に問い合わせを行う.

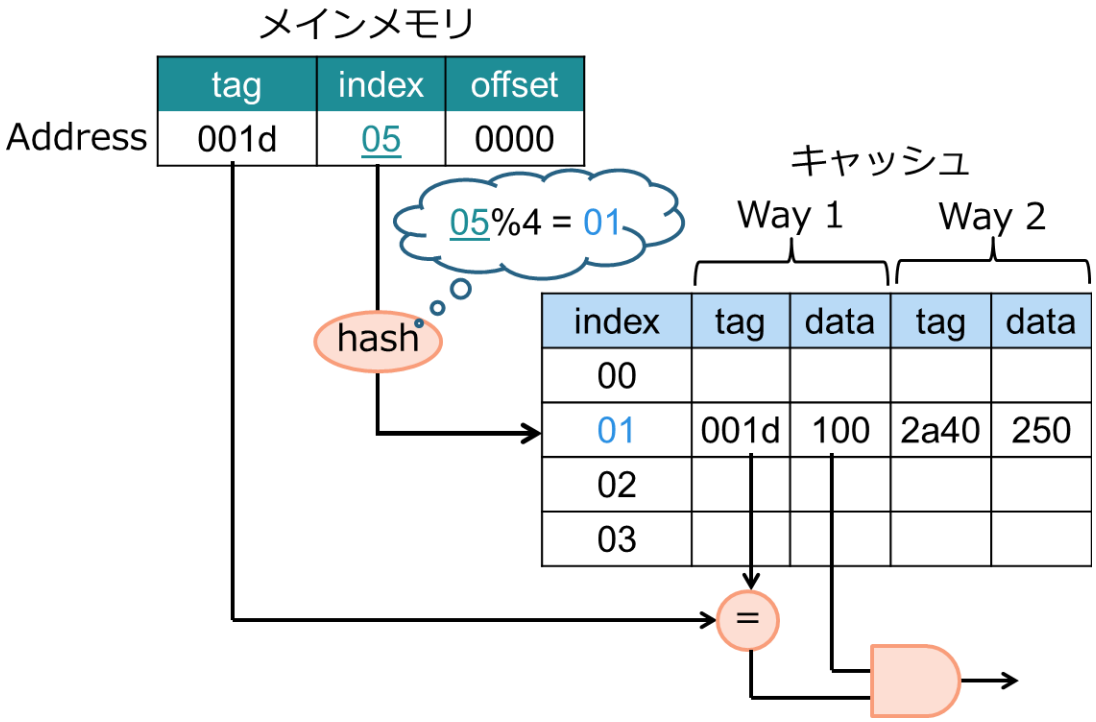

図 2.3　メインメモリからキャッシュメモリへの格納先の読み替え動作

2.3. キャッシュミス

本節では，キャッシュミスとその種別について解説する．プログラムの実行時に要求されたデータがキャッシュに存在すればキャッシュヒット，存在しなければキャッシュミスとなる．キャッシュヒットが多ければ多いほどプログラムの実行効率，すなわちプロセッサの性能は高まり，ミスが多ければ多いほど実行効率は下がる．

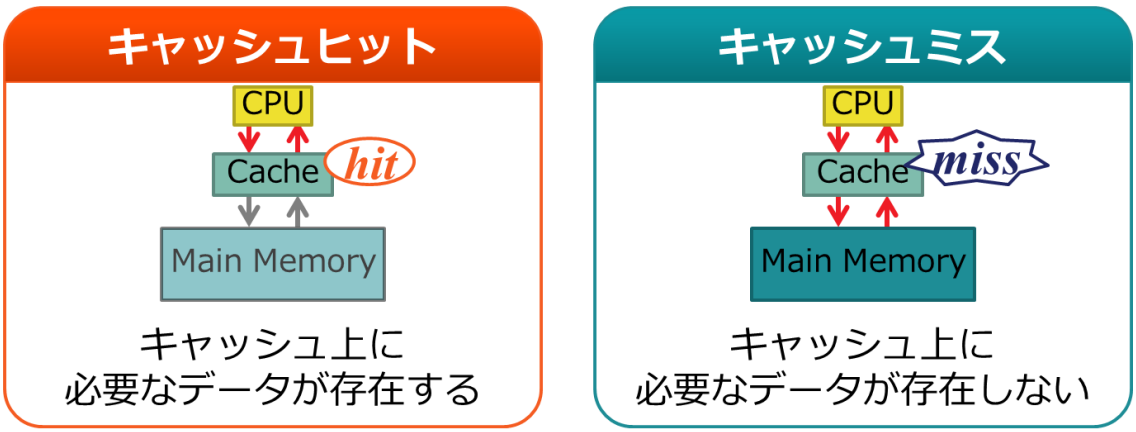

図 2.4　キャッシュヒットとキャッシュミス

2.3.1.　初期参照ミス

初期参照ミスは，あるアドレスへのアクセスが初めてであったときに，該当データがキャッシュに乗っていないがために起こるキャッシュミス．初期参照ミスを起こすことでそのアドレスの該当データがキャッシュに乗り，2 回目以降のアクセス，再参照時にキャッシュヒットするようになる．

これはプログラム内蔵方式で動作するプロセッサとして，回避することが不可能なキャッシュミスである．ストリームアクセス（3.1.2 節に後述）など，規則性を持ったアクセスであれば，初期参照であったとしてもプリフェッチ（3.1.2 節に後述）によりそのミスによるレイテンシを隠蔽できるが，予測が難しいアクセスパターンであれば，初期参照ミスは必ず被ることとなるミスの 1 つである．

初期参照ミスに対して，あるアドレスへのアクセスが二度目以降であるにもかかわらず，既に該当データがキャッシュから追い出されているために起きたキャッシュミスを再参照ミスと呼ぶ．その実態は，以下の競合性ミスや，容量性ミスである．

2.3.2. 競合性ミス

複数のキャッシュラインが，その数よりもウェイ数が小さなひとつのキャッシュセットを巡って競合する場合に発生するキャッシュミス．キャッシュ全体では容量に余裕があるにもかかわらず，アドレスのハッシュ結果の偏りにより特定のセットにアクセスが集中することで追い出しが発生し，キャッシュミスとなる．

2.3.3. 容量性ミス

ワークロードの実行に使用されるアドレスの数が，そのキャッシュメモリの持つインデックスで用意できるセットの数を超えたことで起こるキャッシュミス．キャッシュメモリの容量を一定のままインデックス数を増やすにはウェイ数を減らす必要があり，トレードオフの関係となる．

2.3.2 節で述べた競合性ミスについても，1 セットあたりのウェイ数を増やすことで防ぐことができるミスなので，容量性ミスの一種とみなすことができる．

2.4. キャッシュ構成の評価の指標

本論文で，キャッシュアルゴリズムごとの性能を計る指標として，MPKI と IPC を使用する．それぞれ以下の節で解説する．

2.4.1. MPKI

MPKI は Miss per kilo instructions の略で，1000 命令実行あたりに何度キャッシュミスを起こしたかを意味するキャッシュ性能評価の指標．キャッシュアルゴリズムにより変動するのはもちろんのこと，ベースラインとなる MPKI はワークロードごとにも異なる．一般的に広い範囲でのストリームアクセス（3.1.2 節に後述）を行うワークロードほどミス数が増え，それに連れて MPKI も増大してしまう．また，キャッシュアルゴリズムだけでなく，キャッシュメモリに使用する記憶デバイスのレイテンシ，パイプライン段数，発行幅などによっても変動する．

MPKI の計測として，ワークロードの実行中の特定の箇所での 1000 命令の間のミス数を示す場合もあるが，本論文ではキャッシュミス回数÷全実行命令数×1000 の計算で MPKI を示す．

ミスを減らせることは命令実行の効率を高めることに繋がるので，一般的に MPKI は低ければ低いほど良い．ただし，プリフェッチ（3.1.2 節に後述）により積極的にキ

ャッシュ容量を活用し投機的なキャッシュ読み込みのためのミスをすることで，MPKI は向上するが IPC も向上するというケースもある．

2.4.2. IPC

IPC は Instruction per cycle の略で，プロセッサの動作クロック 1 サイクルあたりに何命令実行できるかを意味するプロセッサ性能の指標．キャッシュとの関係としては，キャッシュのヒット率が高ければ高いほどパイプラインがデータ待ちで止まる可能性が低くなり，IPC が上がることで円滑にプログラムが実行される．これに伴い，プロセッサのハードウェア・演算器の資源利用率も高まる．逆に，キャッシュミスが多ければ多いほど，パイプラインへのデータ供給に時間がかかり，IPC が下がる．これに伴い，プログラム実行に要する時間も延び，パイプラインの資源利用率も低下する．

第3章 関連研究

この章では，既存のキャッシュアルゴリズムについて述べる．キャッシュアルゴリズムのなかで特に重要な 2 つのテクニックとして，置き換えアルゴリズムとプリフェッチが存在する．

3.1. キャッシュアルゴリズム

3.1.1. 置き換えアルゴリズム

置き換えアルゴリズムは，キャッシュに残されるべきデータを，各キャッシュライン重要度の判断するために，アクセス履歴による判断し，時間局所性のあるプログラムのキャッシュアクセスを支援する．代表的なキャッシュ置き換えアルゴリズムに，LRU がある．このアルゴリズムでは，セットの中で最も昔にアクセスされたキャッシュラインが最も再参照されにくいと見なして，新たなラインの挿入時の追い出し対象とする．これにより，時間的局所性のある多くのプログラムに都合の良いキャッシュラインメンテナンスが行われる．

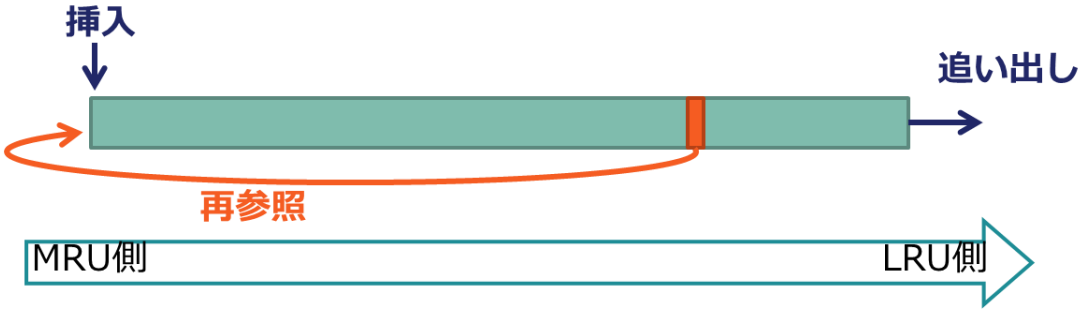

図 3.1 LRU アルゴリズムにおける追い出し順序の操作

本論文において提案機構を評価する比較対象として，ベースラインを LRU とする．その条件設定については第 6 章，第 8 章でも再度より詳しく述べる．また，提案機構を実装する構成において，特に断りのない場合，提案機構は LLC にのみ適用し，LLC 以上の階層のキャッシュの置き換えアルゴリズムは LRU とする．

3.1.2. プリフェッチ

プリフェッチ[4,7,9]は，初期参照ミスを削減することを目的としたキャッシュアル

ゴリズムである．プリフェッチについて述べる前に，プリフェッチが対象とするストリームアクセスについて説明する．

ストリームアクセスとは，連続したメモリ空間へのアクセスを行うようなアクセスパターンを指す．具体例を次の図 3.2 に示す．横軸の時間軸の経過に合わせて決まったアドレス軸方向右上に向かって伸びている直線状のアクセスがそれである．

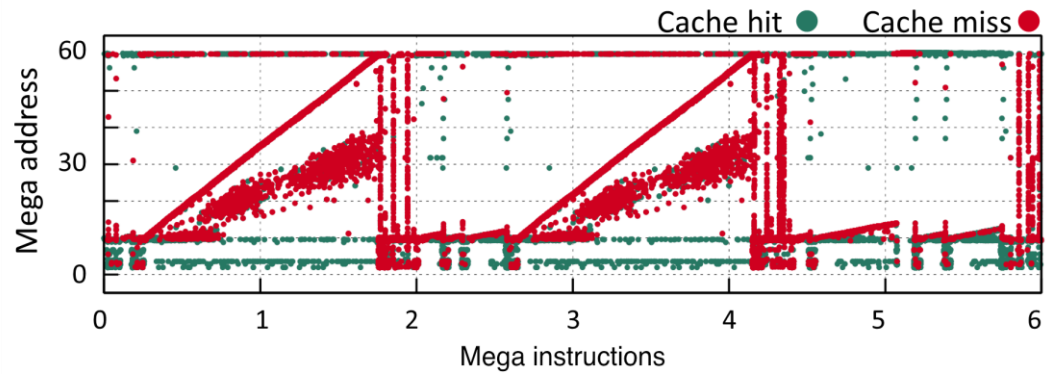

図 3.2　ストリームアクセスの実例（483.xalancbmk）

ストリームプリフェッチは，このストリームアクセスを予知可能なアクセスと見なし，プログラムがそのアドレスを必要として要求する前に予めキャッシュに載せておくという手法である．このプリフェッチを行う機構をプリフェッチャという．

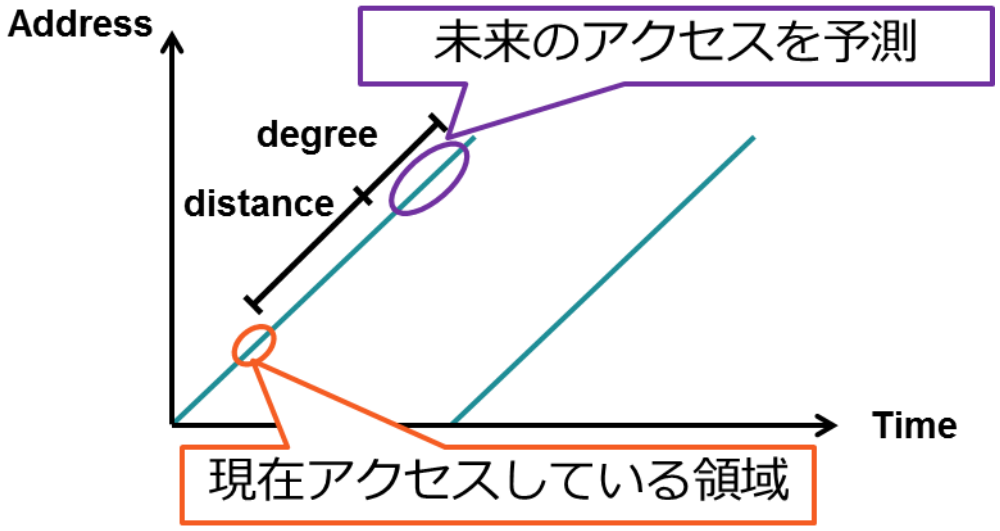

図 3.3　ストリームアクセスとストリームプリフェッチャの働き

また，ストリームプリフェッチの変形としてストライドプリフェッチが存在する．ストライドアクセスというのは，図 3.4 中で示されるように，規則的かつ断片的なア

クセスのことである．このアクセスを予測して，前もってキャッシュに載せるのがストライドプリフェッチである．

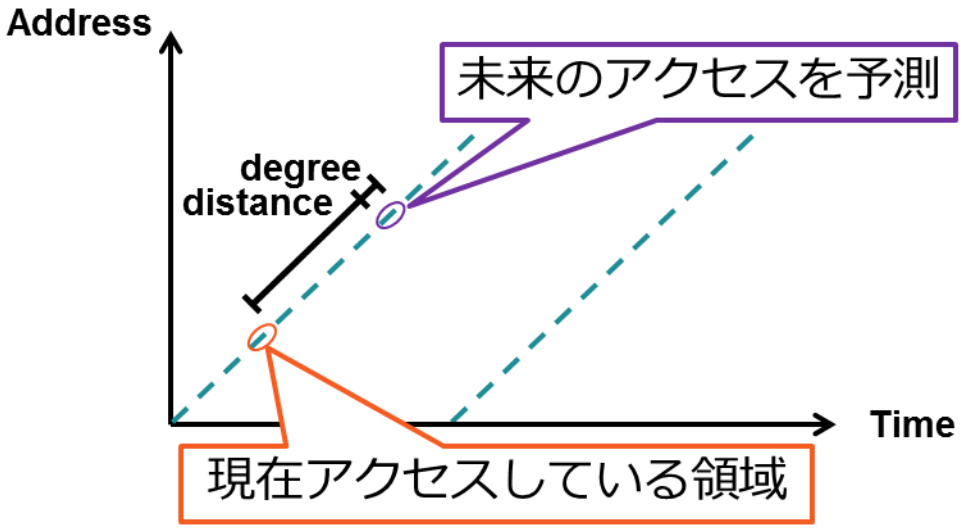

図 3.4　ストライドアクセスとストライドプリフェッチャの働き

3.1.3.　プリフェッチと協調する置き換えアルゴリズム

プリフェッチにより，キャッシュ容量を積極的に活用することでプロセッサの性能向上を狙う．しかしながら，置き換えアルゴリズムによっては，再参照のためのデータ保持をプリフェッチによりすべて流し去られてしまう場合がある．

LRU は置き換えアルゴリズムとして比較的シンプルで低コストにミス削減において一定の効果を上げることが知られている[16]．しかしながら，LRU とプリフェッチの組み合わせを考えたときに，LRU はその動作の単純さ故に，重要でないデータであっても一度 MRU 側に置いてしまい，より重要なデータを LRU 側に押し出してしまうような場合である．つまり，プリフェッチ直後の競合性の追い出しによるミスである．例えば，1 度しか使用されないようなストリームアクセスで使用するキャッシュラインが，プリフェッチによって次々とキャッシュに載せられることで，それまでに同じセットに存在した，再参照される可能性のあるラインが追い出されてしまう．

この問題の解決には，従来のキャッシュアルゴリズムではプリフェッチャと協調する置き換えアルゴリズムを使用するという手段が執れる．例えば，RRIP [3], PACMan [17]がそれにあたる．

Jaleel らが提案する Re-Reference Interval Prediction (RRIP)は，キャッシュへの挿入時に，MRU ではなくある程度 LRU 側に挿入することで，再参照があったラインのみが LRU 側に移動する．これにより，再参照されやすいラインが，たくさんの時間局所的に再参照されないラインに追い出されることを防いでいる．

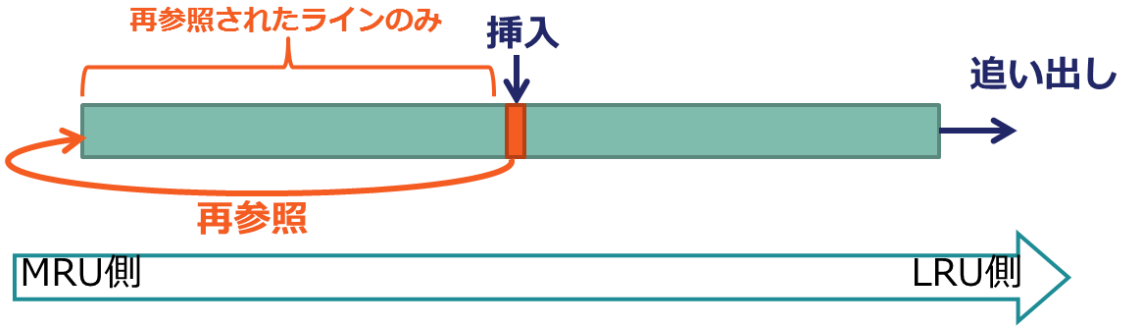

図 3.5　RRIP アルゴリズムにおける追い出し順序の操作

Wu らの提案する PACMan [17]は，RRIP をベースとした手法で，プリフェッチによりキャッシュに挿入されるラインのみを LRU 側に挿入することを特徴とする．つまりプリフェッチによる挿入のみを冷遇して，デマンドのアクセスは LRU 側に優遇している．

RRIP と PACMan のどちらの置き換えアルゴリズムも，プリフェッチによってキャッシュに載せられたデータを冷遇することから，Prefetch aware な置き換えアルゴリズムと呼ばれる．これらのアルゴリズムは，プリフェッチによるアクセスが一般に時間的局所性が薄いことを理由に，LRU 側におき，キャッシュ中のデッドブロックの発生を防いでいる．このような手法とは逆に，プリフェッチによる挿入であろうと MRU 位置に挿入してしまうような LRU を Prefetch friendly な置き換えアルゴリズムと呼ぶ．

また，一方的にプリフェッチにより挿入されるラインを置き換えアルゴリズムで捌くだけでなく，プリフェッチャが置き換えアルゴリズムに歩み寄ってプリフェッチ量を調整する手法[22]が Irie らによって提案されている．

第4章 キャッシュミスの調査

本研究では，既存のキャッシュアルゴリズムを適用してなお救えないキャッシュミスが存在すること，またそれはどういうものであるかを調査した．本章では，その調査結果について議論する． 本章ではまず，第 3 章で触れた関連研究を踏まえて一般的なプロセッサの持つそれらのキャッシュアルゴリズムについて述べる．

次に，我々の調査において注目した従来のキャッシュアルゴリズムで救えないキャッシュミスの一つとして，本研究では長期再参照ミスについて議論する．その定義と，長期再参照ミスを減らすことのモチベーションについて述べる．

4.1. 既存のキャッシュアルゴリズムで救えていないミス

一般的なプロセッサは，その構成要素としてプリフェッチ機構とスマートな追い出しアルゴリズムを持つ．より具体的には，Intel Pentium4 [5]，IBM POWER4 [6]以降の，特にハイエンドな位置づけにあるプロセッサアーキテクチャにおいて採用され，動作している．特に，プリフェッチは，半導体プロセスの微細化の恩恵を得た LLC の大容量化との相性もよく，これら以降の世代のプロセッサが性能を大きく伸ばした要因の一つとなっている[8]．

このように，プリフェッチが LLC のキャッシュ容量を有効活用することで性能向上をもたらし，スマートな追い出しアルゴリズムがプリフェッチによるデッドブロックから保護すべき価値のあるデータをキャッシュに残す仕組みが現在のキャッシュシステムに高い性能を生み出している（図 4.1）．

しかしながら，3.1.2 節にて後述するように，プリフェッチをもってストリームアクセスの初期参照ミスを救い，置き換えアルゴリズムによってプリフェッチによるキャッシュ領域の汚染を緩和させてなお生じるキャッシュミスが存在する．本研究では，初期調査として，そのようなキャッシュミスがどのようなものであるのか調査した．

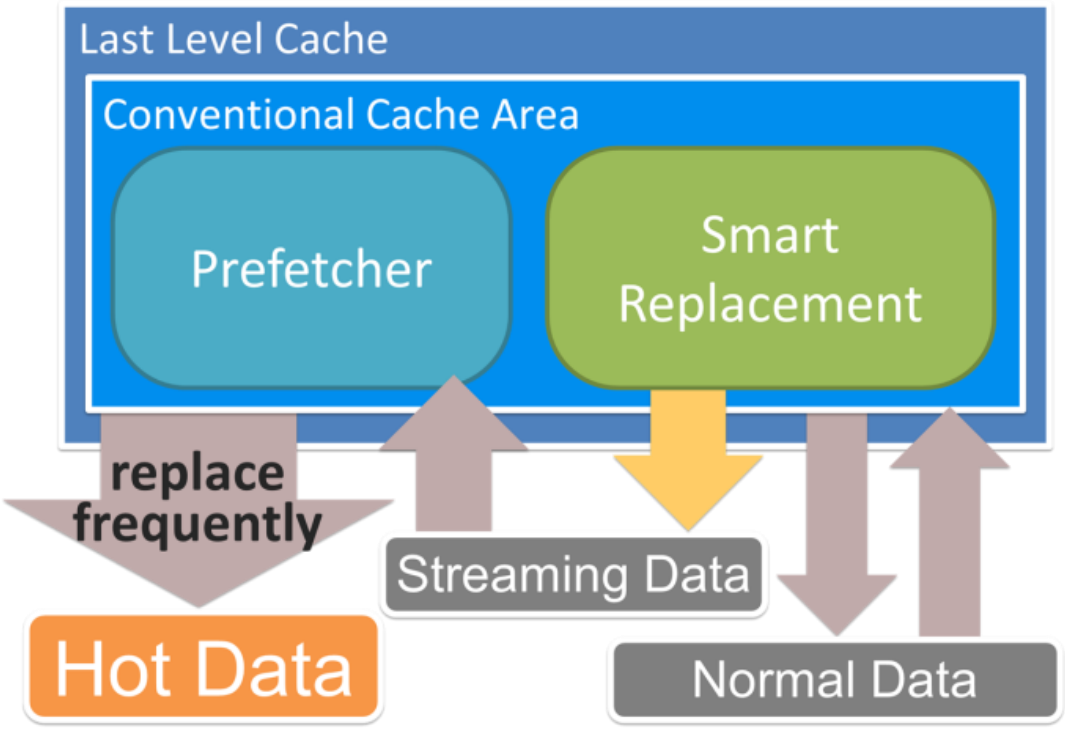

図 4.1　従来のキャッシュシステム

4.2.　長期再参照ミス

本節では，我々の調査の過程において発見した，本研究における重要な知見である長期再参照ミスについて述べる．これは，既存のキャッシュアルゴリズムで救えないキャッシュミスの一つであり．それは 1M 命令以上の距離がある再参照時のミスによるものであった．

このような，一度アクセスされたあるアドレスに対応するラインがキャッシュに載った後，実行命令数が 1M 命令以上経過した後に再度アクセスされたケースを本研究では**長期再参照**として定義する．また，長期再参照を生じるまでに，該当のキャッシュラインが追い出されていたためにミスが発生したケースを，本研究では**長期再参照ミス**と呼ぶ．

4.3.　長期再参照ミスの実測例

再参照距離とミス数の関係を，実測例を挙げて解説する．図 4.2 は予備評価のプロセッサシミュレーション結果から，再参照の距離ごとのミス数を示したものである．

この予備評価は，SPEC CPU 2006 Benchmark Suite [2,15]から数本のベンチマークを使用してキャッシュの挙動を観察したものである．図 4.2 はそのなかから代表的な 471. omnetpp での動作から確認できたキャッシュアクセス傾向による．測定環境は Alpha ISA [23,24]のサイクルアキュレートなプロセッサシミュレータ鬼斬弐[18]で，このときのキャッシュアルゴリズムは L1，L2，L3 共に LRU．キャッシュ容量は DL1，IL1 が 64KB， L2 が 512KB，L3 が 2MB である．Srinath らによるストリームプリフェッチャ[4]を有効にしている．なお，ここでのキャッシュミス数カウントでは，プリフェッチによるミスは除外している．なぜなら，プリフェッチはその特性から，本来ミスをするタイミングより前もってプリフェッチを行うために投機的にミスを発生させ，ミスによるレイテンシを隠蔽しているため，性能向上を阻害するミスではないためと，プリフェッチによるミスはそれ以上回避しようのないミスであるためである．

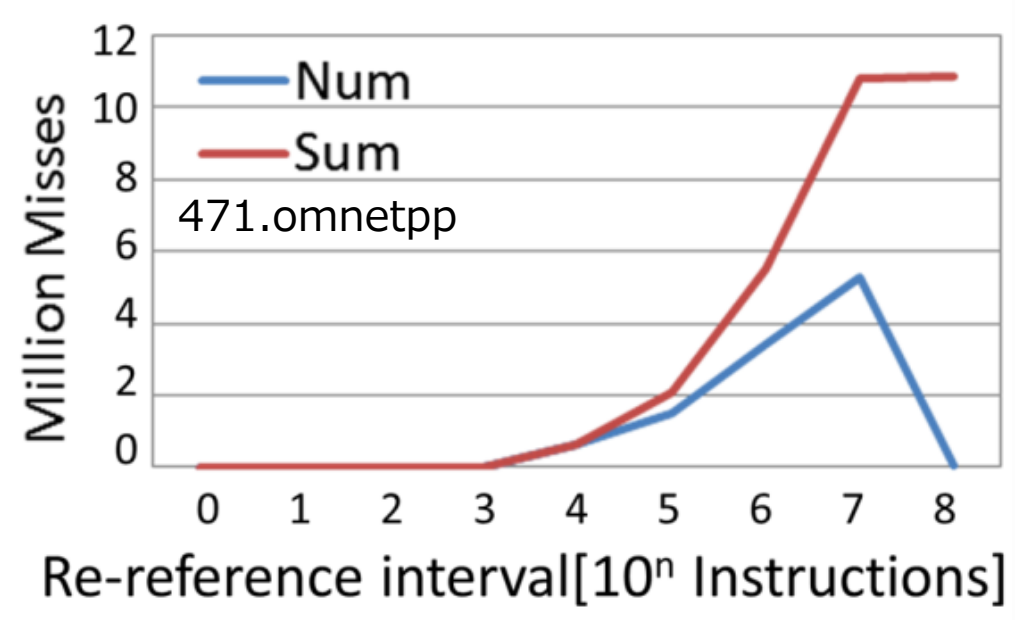

図 4.2 再参照距離とミス数

このグラフは, 横軸が再参照距離を, 実行命令数を基準として対数オーダーで示し, 縦軸がそれぞれの再参照距離オーダーでのキャッシュミス数を示している．青い線はそれぞれの再参照距離でのミス数の絶対値，赤い線は距離 0 から 10^8 命令のオーダーに向かっての累積ミス数である．

このグラフからわかるように，オーダー別でキャッシュミス発生数が最多であるのは 10M 命令オーダーであり，継いで 1M 命令オーダーでの長期再参照ミスが多い．

累積で見ても，100k 命令オーダーから 10M 命令オーダーにかけて増分が大きい．つまり，現存するキャッシュミスの正体の大部分は，長期再参照が占めている．

4.4. 長期再参照ミスの実態

長期再参照ミスの実例として，図 4.3 にベンチマーク 471.omnetpp でのキャッシュアクセス傾向，長期再参照におけるミス発生の様子示した．このグラフの横軸は命令実行数と，それに伴う時間経過をメガ命令オーダーで表示したもので，縦軸はアドレス空間のうち約 40 キロアドレスの範囲を抜粋したものである．プロットされている丸ひとつひとつがあるアドレスへアクセスを示し，緑ならばキャッシュヒット，赤ならばキャッシュミスを起こしていることを示している．

このグラフの範囲において，あるポイントを横軸に左右に動かした箇所にあるポイントは，全て同じアドレスへのアクセスの履歴である．そのことから，あるアドレスではずっとヒットし続けている，また別のアドレスではずっとミスし続けている，さらに別のアドレスではヒットとミスが混ざっていることがわかる．

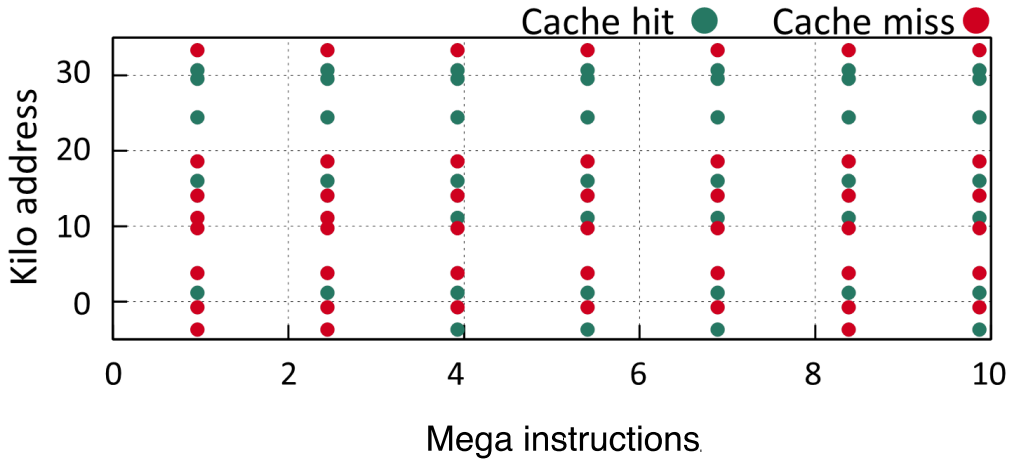

図 4.3　キャッシュアクセス傾向, 長期再参照におけるミス発生の様子 (471.omnetpp)

キャッシュの構造上，同一アドレスへのアクセス，すなわち再参照である場合，そのアドレスのデータを保持するキャッシュラインは追い出されずに保持され続けていることが期待される. しかしながら, このように同一アドレスへのアクセスであっても，2M 命令程度の間隔でアクセスされることでその間隔の間に該当するキャッシュラインが追い出され，キャッシュミスが発生．この再参照のために再度下位のメモリシステム，ここでは L3 からメインメモリへのアクセスを生じさせてしまっている．

ミスを頻発している複数のキャッシュラインに相当するプログラム中のアドレスの

特性は，そのアドレスが属するアドレス空間から判断することができる．

4.5. 長期再参照ミスが占める割合が大きい理由

4.3 節の図 4.2 で示した通り，ここでは LLC でのキャッシュミスのうち，長期再参照ミスが占める割合が大きい理由を考える．

これは，対象が LLC であることに起因する．再参照の距離が短いアクセスのほとんどは，LLC である L3 キャッシュへの問い合わせの前に，実行コアに近い順に L1 キャッシュ，L2 キャッシュでヒットすることで解消している．LLC の視点から見れば，ある意味，L1，L2 によって短期の再参照アクセスがフィルタされているように見えると言ってもよい．L3 には，長期的な再参照であるが故に L1，L2 の容量では収まり切らず，フィルタしきれなかったようなアクセスが流れてくる．

4.6. 長期再参照ミスが起こる理由

長期再参照がミスとなる主な原因は，時間経過に伴う該当キャッシュラインの追い出しである．LRU や RRIP などの従来の置き換えアルゴリズムでは，時間的局所性に焦点を当てており，また，プリフェッチも直近の範囲での空間的局所性に着目して動作する．このように，従来のキャッシュアルゴリズムは直近のアクセス履歴を元に動作し，直近のアクセスを予想してそれを受け入れる体勢をとるような機構である．

4.5 節でも議論したように，短期間での再参照は L1，L2 キャッシュによって救われるものとして，だからこそ L3 では長期の再参照によるミスを救うような仕組みがあれば相互にキャッシュミスの種別をカバーすることができる．しかしながら，従来のキャッシュシステムでは，L1，L2 キャッシュと，L3 キャッシュで明確に保護する対象を変えるような戦略，アルゴリズムの組み合わせは採られておらず，L3 キャッシュであっても短期の再参照を優先的に保護するようなアルゴリズムとなっている．

また別因として，長期再参照であっても一度はその直前にプリフェッチによってキャッシュラインに再度乗ったにも関わらず，そのラインを使用される前に追い出されるケースがある．これは，“該当キャッシュラインへのアクセス→追い出し→プリフェッチによるキャッシュオン→競合による使用前の追い出し→キャッシュミス”，というケースである．これは 2.3.2 節で述べた競合性ミスと同様の現象である．

第5章 Stubborn 戦略の提案

本章では，キャッシュ置き換えの戦略である Stubborn 戦略を新たに提案し，その利点について述べる．

5.1. 長期再参照ミスを救うには

ここまでの章で，これまでのキャッシュアルゴリズムでできることと，できないことについて議論してきた．特に第 4 章では，LLC において対処すべきミスは長期再参照ミスであり，従来のキャッシュアルゴリズムでは長期の再参照をターゲットと見ていないこと，そのために救えていない長期の再参照によるミスがあることを明らかにした．

従来のキャッシュアルゴリズムは直近のアクセス履歴を元に動作し，直後のアクセスを予想してそれを受け入れる体勢をとるような機構である．しかしながら，長期再参照ミスを起こすようなアクセスパターンは直近のアクセス履歴からは推測できない．これが長期の再参照によるミスを救うことができない理由である．

では，長期再参照ミスを救うにはどのような戦略を持って挑めば良いか，その最も単純明快な答えは，直近のアクセス履歴に応じて細かい粒度でキャッシュを入れ替えるようなことはせず，長期再参照ミスを起こすキャッシュラインを，そのプログラムフェーズが続く限り追い出さずにずっと持ち続ければよい，というものである．

5.2. Stubborn 戦略

これまでの長期再参照ミスの特長の調査と議論を踏まえて，本研究では，ある基準によって定めたキャッシュラインを**長期間追い出さない**ことを特長としたキャッシュアルゴリズムの基本戦略を提案する．我々はこれに Stubborn 戦略と名付けた．“Stubborn”は“頑固”を意味する．

Stubborn 戦略は，キャッシュ中に追い出しを起こさない領域をつくることで実現させる．（図 5.1）

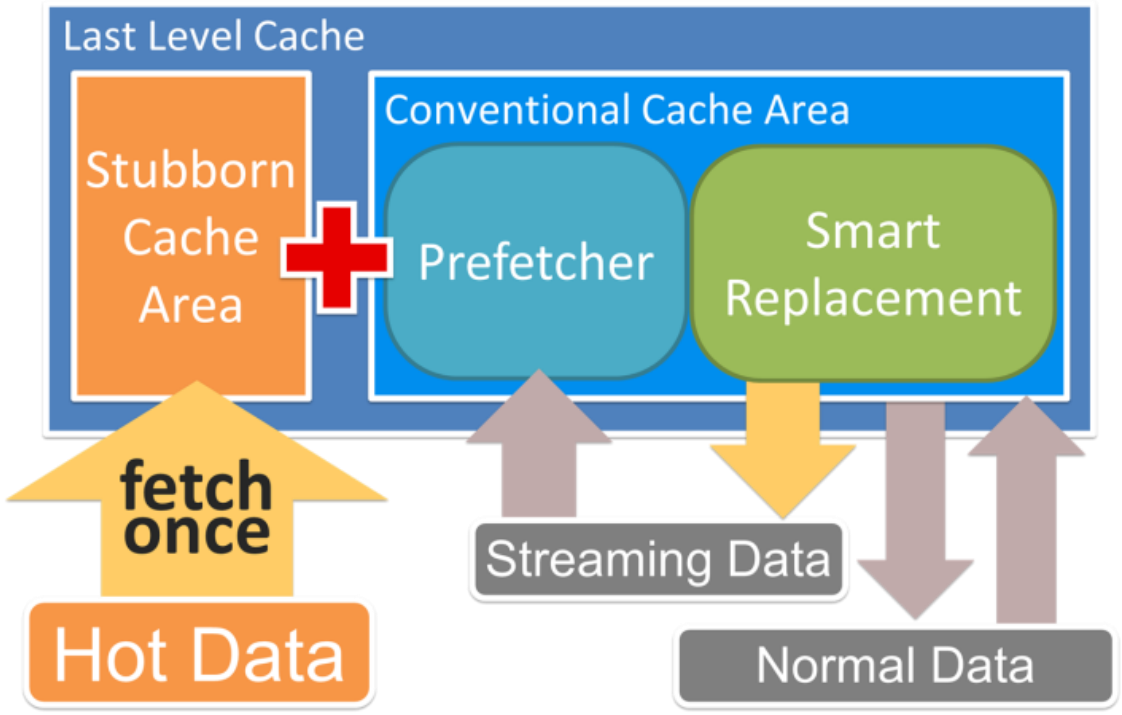

図 5.1　Stubborn キャッシュ領域をもつ提案キャッシュシステム

5.2.1.　追い出し不可属性

定めたキャッシュラインを追い出さない仕組みを作るため，置き換えアルゴリズムに**追い出し不可属性**を新たに導入する．この追い出し不可属性はキャシュラインの単位で付与され，この属性の付いたキャッシュラインはキャッシュから追い出されなくなる．

図 5.2 に追い出し不可属性を持たせたキャッシュセットの置き換え動作のイメージ図を示す．キャッシュラインへの挿入後，あるタイミングで追い出し不可属性が付与された複数のキャッシュラインは常に MRU 側にあり，追い出されることはない．追い出し不可属性が付加されていない通常のキャッシュラインは，他のキャッシュラインの再参照や挿入により，LRU 側へ遷移していき，最終的には追い出される．挿入は，追い出し不可属性を持っているキャッシュライン達よりは追い出し側，LRU 側の中で，通常のキャッシュライン達の中では最も MRU 側の位置に挿入される．通常ラインの再参照時にも同様の位置に移動することになる．この動作は，1 セットの中での Stubborn 領域が最小単位しかない場合でも，半分でも，全てであっても成立する．1 セットが全て Stubborn 領域であり，追い出し不可属性が付与される場合は，そもそも挿入と追い出し順序の整理は不要となる．

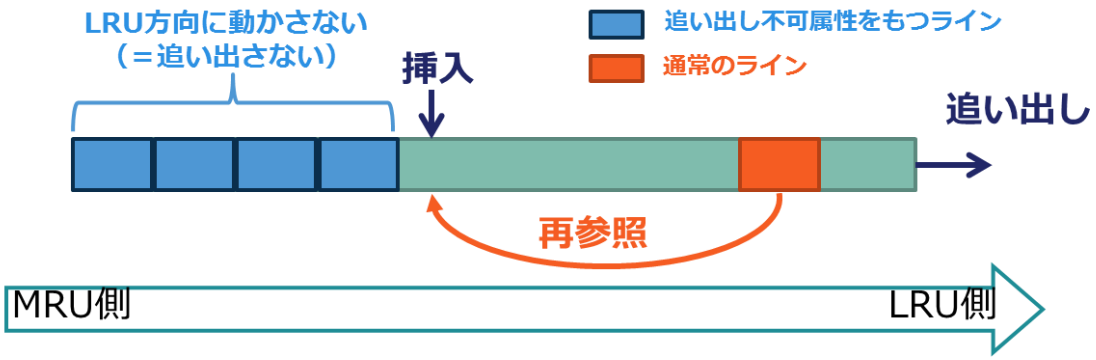

図 5.2　追い出し不可属性と置き換え動作

なお，図 5.2 では従来の置き換えアルゴリズムとの比較のために，単一キャッシュテーブルの同一キャッシュライン中に Stubborn 戦略を適用し追い出しをしない Stubborn 領域と，通常の領域を混在させているが，Stubborn 戦略に基づくキャッシュテーブルを LLC のキャッシュテーブルとは別に設けて実現することもできる．後述の第 6 章でのポテンシャル評価では，そのような構成となっている．

5.2.2.　Stubborn 戦略により長期再参照を救える理由

一度追い出し不可属性を設定したキャシュラインは，その後追い出されることはない．すなわち，そのラインへのアクセスは永続的にヒットし続けることとなる．この特性は，長期再参照に対して次の 2 パターンのメリットを発揮する．ひとつは，ストリームアクセスのような予測可能なアクセスパターンでなくても，再参照時に必ずそのキャッシュラインへのアクセスをヒットさせられること．もう一つは，キャッシュ中の一定領域を変動させずに取り置くことで，玉突き的な競合ミスの繰り返しにより特定のセットでミスし続けるラインの割合を抑えることができることにある．

この結果として，この Stubborn 戦略によって，長期再参照ミスを救うことができる．

5.2.3.　追い出し不可属性付与アルゴリズムの実現方法

Stubborn 戦略を実現するキャッシュをつくるにあたって，追い出し不可属性を付与する対象のキャッシュラインを選択するアルゴリズムが必要となる．Stubborn 戦略によるキャッシュミス削減の効果は，この選定アルゴリズムがいかに救うべきキャッシュラインを拾えるかどうかで決まる．以下の節に本研究での選定アルゴリズムの案を挙げる．

5.2.3.1. 先着順採用

追い出し不可属性の付与を開始してから，先着順に各キャッシュセットに格納されたキャッシュラインに追い出し不可属性を付与する方法．追い出し不可属性を付加したラインの数をカウンティングし，一定数に達するとそれ以降は追い出し不可属性の付加をしなくなる．これは選択アルゴリズムの中でも最も単純な方法である．

5.2.3.2. ランダム採用

一定の，あるいは可変の確率で各キャッシュラインの追い出し不可属性の採択を決定する方法. 先着順採用と比べて, 採択するラインがワークロード中に広く分散する．これにより，採用されるキャッシュラインが，採用開始タイミングに依存した特定のアドレス範囲に集中するのを避けることができる．ランダム採用ではあるが，採用条件は乱数で当たるというほかに，セットあたりの追い出し不可属性を付加するラインの数に達していないか，がある．

また，先着順採用と比べて，Stubborn 領域が追い出し不可属性を付与したラインで満たされるまでの所要時間が延びる．採択確率とワークロードによっては，所定の命令数実行を終えても Stubborn 領域が埋まっていないことが起きうる．

ランダムで確率的に採用をしているが，命令実行期間ごとに平均して採用するわけではないので，ある程度は先着順採用と同様の傾向が見られることとなる．

5.2.3.3. 実行履歴を利用した採用

Stubborn 戦略における追い出し不可属性の付加アルゴリズムの特長として，付加するかどうかの判断を，追い出し直前まで保留することができることが挙げられる．上記の先着順採用，ランダム採用では，挿入時に追い出し不可属性を付加するかどうかを判断しているが，本来はそのキャシュラインが LRU オーダーのなかで初めて追い出し対象として追い出すかどうかを判断される位置まで移動してしまった瞬間までは，追い出し不可属性を付けるかの判断を遅延することができる．

その遅延判断までの間，RRIP における RRPV のようなカウンティングの追加ハードウェアを投じることで，実行履歴を利用して追い出し不可属性を付加するかどうか判断することができる．

また，ここまでに挙げた選択手法は相互に排他ではなく，組み合わせて，あるいはアダプティブに切り替えて運用することもできる．

第6章 Stubborn 戦略のポテンシャル評価

Stubborn 戦略が最も効果的に利用できた場合に，どれだけの効果が得られるかを予備評価した．本章では理想的な追い出し付加属性の付与の判断基準と，それを使用した Stubborn 戦略を採用したキャッシュシステムの性能評価，考察について述べる．

6.1. 理想的な追い出し不可属性付与

プロセッサシミュレータを用いて以下の手順で研究を進め，Stubborn 戦略のポテンシャルを実証した．

1. 従来のキャッシュシステムを模したプロセッサシミュレータにおける命令実行手順の実行プロファイルを取得し，評価のベースラインとする．
2. 上記プロセッサシミュレータが出力した実行プロファイルを読み込み，提案キャッシュシステムによってキャッシュされるデータを取り扱った場合のキャッシュヒット･ミスを判定する簡易シミュレータによりベースラインとの差をもって評価する．

Stubborn 戦略のポテンシャルを測るために，ここでの評価においてはハードウェア実現性を気にせずに理想的な条件を用意する．具体的には，提案キャッシュシステムに模したシミュレータでは，Stubborn 戦略に基づき追い出しを起こさないキャッシュラインを，過去の実行ログから理想的に選定し，キャッシュライン追い出し不可属性を付与している．そこでは，キャッシュに残し続けるとミス削減が大きくなることがわかっているキャッシュラインを，過去に実行した，評価時と同じ 1G 命令実行範囲の実行履歴から選んでいる．これにより，理想的なキャッシュラインの選択ができていることとなる．

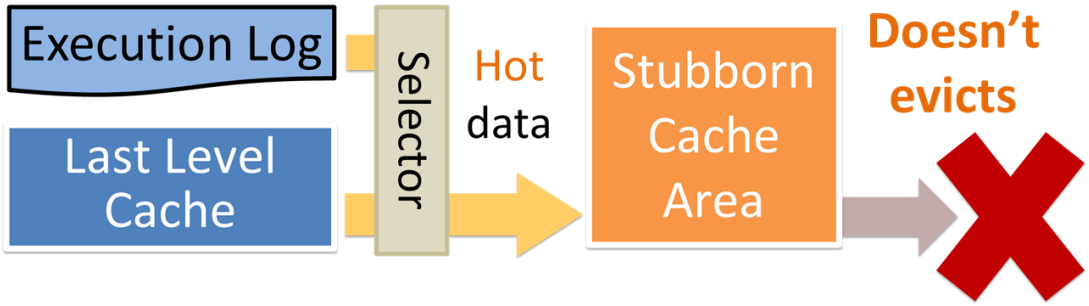

図 6.1 実行ログを元にした追い出し不可属性付与選択

6.2. ポテンシャル評価

6.2.1. 評価環境

ここでは，ベースラインとするキャッシュシステムをもつプロセッサと，提案キャッシュシステムを持つプロセッサを模したシミュレータ上で，実用的な複雑さを持つ様々なアプリケーションをワークロードとしたベンチマークセット SPEC CPU 2006 Benchmark Suite から，ストリームアクセスを持つワークロードを 6 つ抽出し評価した．通常のキャッシュ領域の置き換えアルゴリズムには LRU を使用する． LRU による LLC 容量は 2MB，Stubborn キャッシュ容量は 64KB であり，LLC と同階層に LLC とは別のキャッシュテーブルとして用意した．プリフェッチも掛けており，Srinath らによるストリームプリフェッチャ[4]を L3 で有効にしている．

Stubborn キャッシュエリアには Stubborn 戦略に基づき，追い出しを起こさないキャッシュラインを実行プロファイルから予め定めて格納し，“追い出しを起こさない置き換えアルゴリズム”を適用した．評価の公平を期すため，Stubborn キャッシュへのアクセスで，該当アドレスが格納されていた場合でも，初回の参照はミスとして取り扱った．

6.2.2. MPKI による評価

MPKI による評価結果を次の図 6.2 に示す．

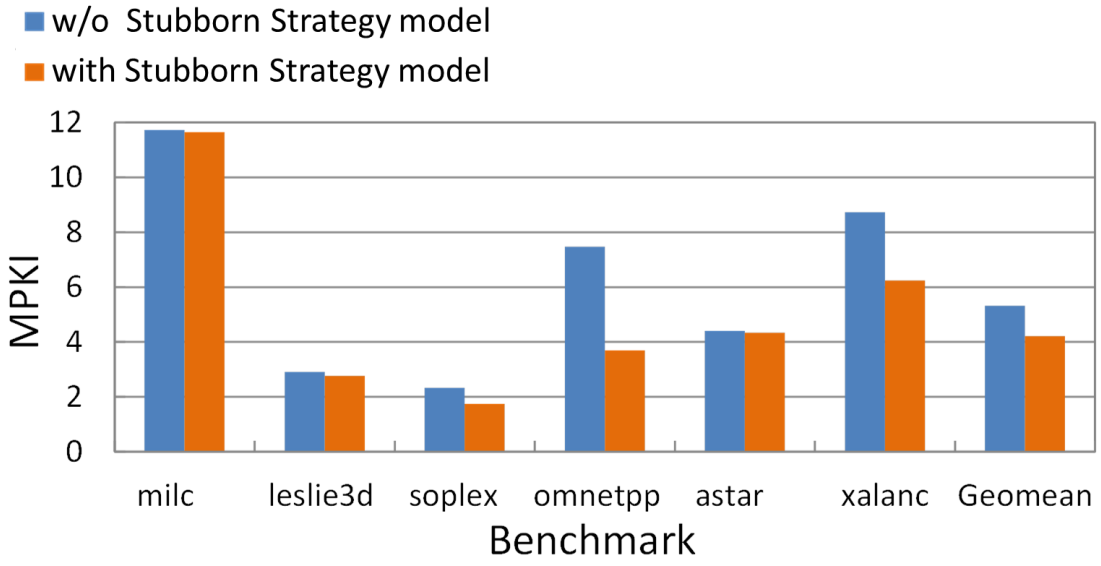

図 6.2 理想的なキャッシュライン選択による MPKI 評価

図 6.2 のグラフでは，青い棒グラフがベースライン，オレンジの棒グラフが Stubborn キャッシュを伴う提案手法の MPKI を示す．この結果より，Stubborn キャッシュを持つシステムが，ベースラインに対して MPKI を大きく引き下げていることがわかる．従来のキャッシュシステムに対してベンチマークによって最大 50.6%のキャッシュミス削減．平均でも 20.9%のキャッシュミス削減が得られ，提案キャッシュシステムである Stubborn 戦略が持つポテンシャルを確認した．

この評価結果は，本研究において提案手法を実現可能なアルゴリズムとした上で目指すべき最高性能となる．

6.2.3. 長期再参照ミスが解消された例

図 6.3 に，Stubborn 戦略に基づくキャッシュを適用する前と後のキャッシュアクセスパターンの変化を示す．グラフの見方は上下共に図 4.3 と同様である．

このプロットから見て取れるように，元々赤い点でキャッシュミスだったアクセスが，緑のキャッシュヒットに切り替わっていることがわかる．また，このグラフの範囲において，あるポイントを横軸に左右に動かした箇所にあるポイントは，全て同じアドレスへのアクセスであり，ミスからヒットに転じたアドレスへのアクセスは時間を問わず全てヒットに転じていることがわかる．これは，全命令実行の 1G 命令の間，該当キャシュラインについて一度も置き換えを起こさないからこそ実現できる．この点は，従来のキャッシュアルゴリズムとの明確な差である．

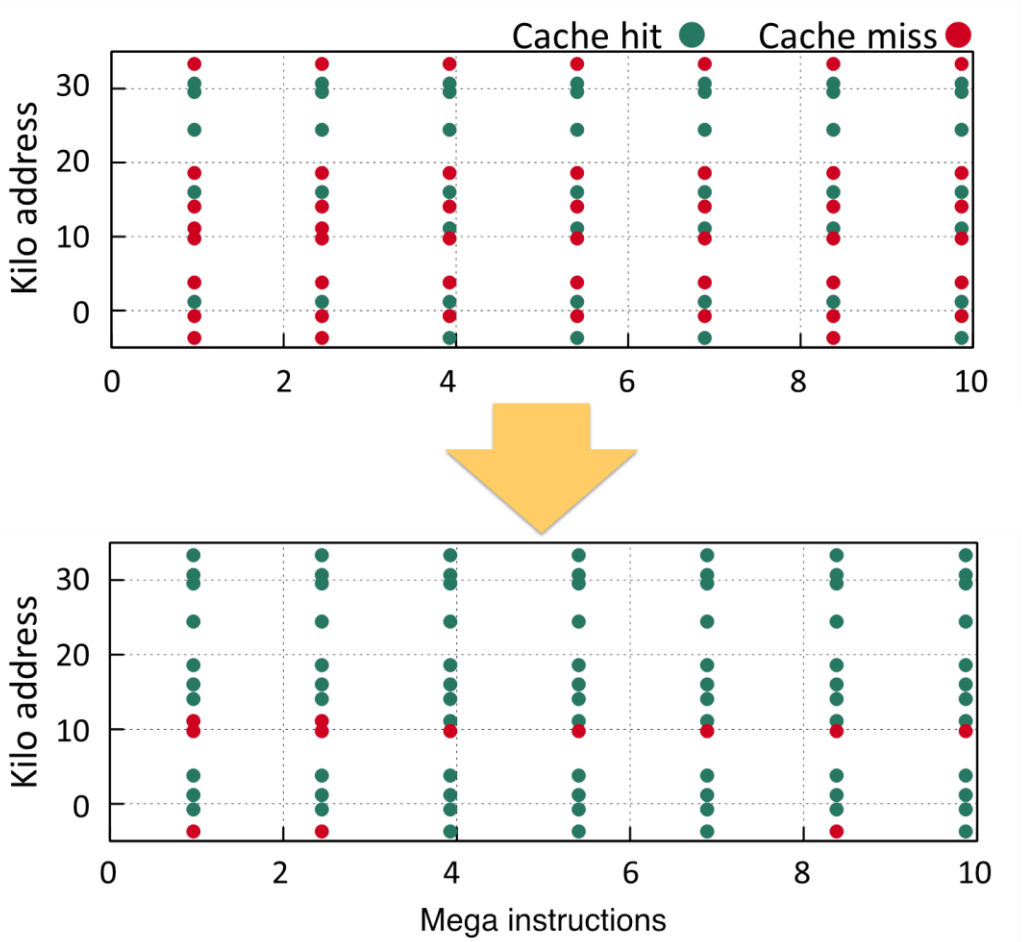

図 6.3　Stubborn 戦略によるアクセスパターンとキャッシュヒット／ミスの変化

第7章 実装

第 6 章では, 理想的な追い出し不可属性付与対象選択アルゴリズムにより Stubborn 戦略のもつポテンシャルを確認した．次に，ハードウェア実現可能な現実的な追い出し不可属性付与対象選択アルゴリズムと，置き換えアルゴリズムとしての具体的な挙動，フラグ操作に関する実装を行い，より正確な Stubborn 戦略の評価を行うための実装を行う．ここでの実装の解説は，サイクルアキュレートなプロセッサシミュレータ鬼斬弐上での追加置き換えアルゴリズムの実装に基づくものである．

7.1. LRU ベースの Stubborn 領域を持つキャッシュ

LRU をベースとした Stubborn 領域をもつキャッシュの実装について述べる．図 7.1 に追い出し不可属性を持たせたキャッシュセットの置き換え動作の実装イメージ図を示す．キャッシュラインへの挿入タイミングで追い出し不可属性が付与された複数のキャッシュラインは，最 LRU 側まで来ても追い出されることはない．追い出し不可属性が付加されていない通常のキャッシュラインは，他のキャッシュラインの再参照や挿入により，LRU 側へ遷移していき，最終的には追い出される．追い出されるタイミングは，最 LRU 側の位置にいる場合の他，最 LRU 側にあるキャッシュラインが追い出し不可属性が付与されている場合も含む．新たなキャッシュラインの挿入先は，通常の LRU と同様に，最 MRU の位置である．追い出し不可属性が付与されたキャッシュラインも LRU オーダーの中で LRU 側にシフトされてゆく．その際は追い出し不可属性のフラグも同様にシフトしてゆく．

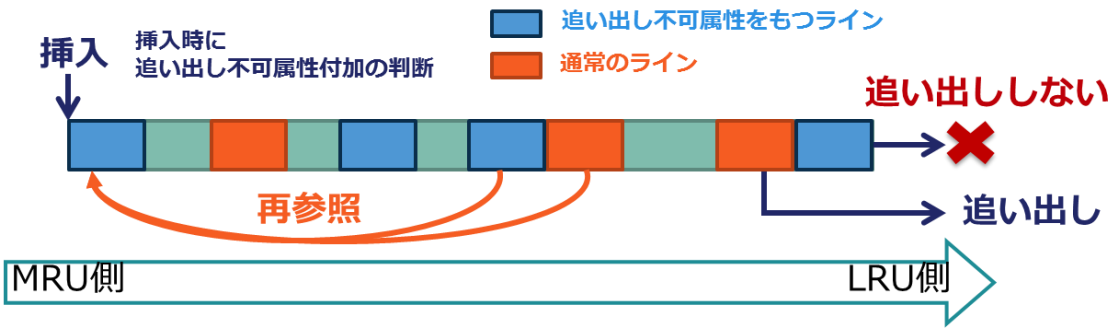

図 7.1 LRU ベースの Stubborn 戦略を適用したキャッシュの置き換え動作

7.2. 追い出し不可属性付与対象選択アルゴリズム

本研究では，追い出し不可属性付与対象選択アルゴリズムの実装として，5.2.3 節で述べた追い出し不可属性付与対象選択アルゴリズムの実現方法のうち，先着順採用とランダム採用のアルゴリズムについて実装した．

7.2.1. 先着順採用

シミュレータにおいて命令実行を開始してから，各キャッシュセットに格納されたキャッシュラインに先着順に追い出し不可属性を付与する．各セットで追い出し不可属性を付加したラインの数をカウンティングし，セット数の半分の数に達するとそれ以降は追い出し不可属性の付加をしなくなる．この実装を Stubborn1.0 と呼ぶ．動作過程を図 7.2 に示す．

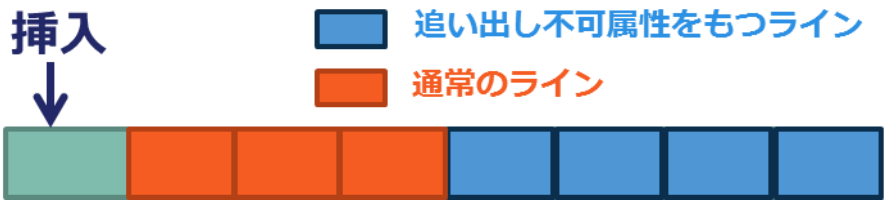

図 7.2 先着順採用での動作過程の例

7.2.2. ランダム採用

一定の確率で各キャッシュラインの追い出し不可属性の採択を決定する．採用条件は乱数で一定確率に当たるというほかに，セットあたりの追い出し不可属性を付加するラインの数に達していないか，がある．

今回の実装では，ランダム採用の手法での採択率を 100%にすることで，先着順採用と完全に同じ挙動が得られる．本論文ではランダム採択率を 50%とする．この実装を，Stubborn0.5 と呼ぶ．動作過程を図 7.3 に示す．

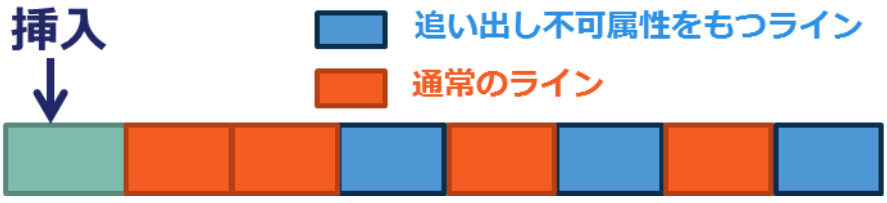

図 7.3 ランダム採用での動作過程の例

第8章 評価環境

8.1. プロセッサ/キャッシュ構成

本章では，前章の実装に基づく現実的な Stubborn 戦略の提案構成の評価を行う環境について述べる．評価環境としたプロセッサシミュレータは鬼斬弐で，Alpha ISA によるサークルアキュレートなシミュレーションを行う．以降の評価における，ベースライン，提案構成に共通するプロセッサアーキテクチャおよびキャッシュシステムの構成を次の表 8.1 に示す．

表 8.1　ベースラインプロセッサ/キャッシュ構成パラメタ

Processor	
Core	Alpha AXP ISA, single core, single thread
Issue width	int:2, fp:2, mem:2
Inst. window	int:32, fp:16, mem:16
Branch pred	8KB g-share
BTB	2K entry, 4way
LSQ	96 entry
Cache Memory	
I/D L1 Cache	LRU, 32 KB, 4 way,
	64 B line, 3 cycle latency
L2 Cache	LRU, 512 KB, 8 way,
	64 B line, 10 cycle latency
L3 Cache (LLC)	Stream prefetcher (degr:16, dist:16)
	2/4MB, 8 way,
	64 B line, 30 cycle latency
Memory access	200 cycle latency

以下の節にて置き換えアルゴリズムとプリフェッチの設定について補足する．

8.1.1. 置き換えアルゴリズム

ストリームプリフェッチを有効として，置き換えアルゴリズムには LRU を使用している．通常のキャッシュ領域の置き換えアルゴリズムには LRU を使用する．Stubborn キャッシュエリアには，LRU ベースで，先着順採用，ランダム採用の 2 種類の追い出し不可属性付与選定アルゴリズムを使用する．これらをそれぞれ Stubborn1.0, Stubborn0.5 と表記する. ランダム採用における採択率は 50%である．

ここで，プリフェッチに関する LRU の挙動として 2 種類の動作パターンを用意した．1 つは Prefetch aware な LRU で，もうひとつは Prefetch friendly な LRU である．その動作の違いは，Prefetch aware な LRU では，プリフェッチしてきたキャッシュラインを LRU オーダーで LRU に置き，Prefetch friendly な LRU では MRU に置くことである．この Prefetch aware な LRU は，PACMan-MH [16]と同様の挙動となる．

8.1.2. プリフェッチャ

ここで使用するプリフェッチャは，Srinath らの論文[4]に記載のストリームプリフェッチャであり, ベースライン, 提案構成共にストリームプリフェッチを有効とした．ストリームプリフェッチは L3 に適用した．

8.2. 評価モデル

上記の構成から， 3 種類の組み合わせでの評価モデルを次の図 8.1 に示す．図からわかるように，提案構成はベースラインのキャッシュテーブルに，さらに Stubborn 戦略に基づく置き換えアルゴリズムを混ぜたものであり，ランダム採用と先着順採用の二種類がある. キャッシュ容量はいずれも平等に 2MB または 4MB で評価している．

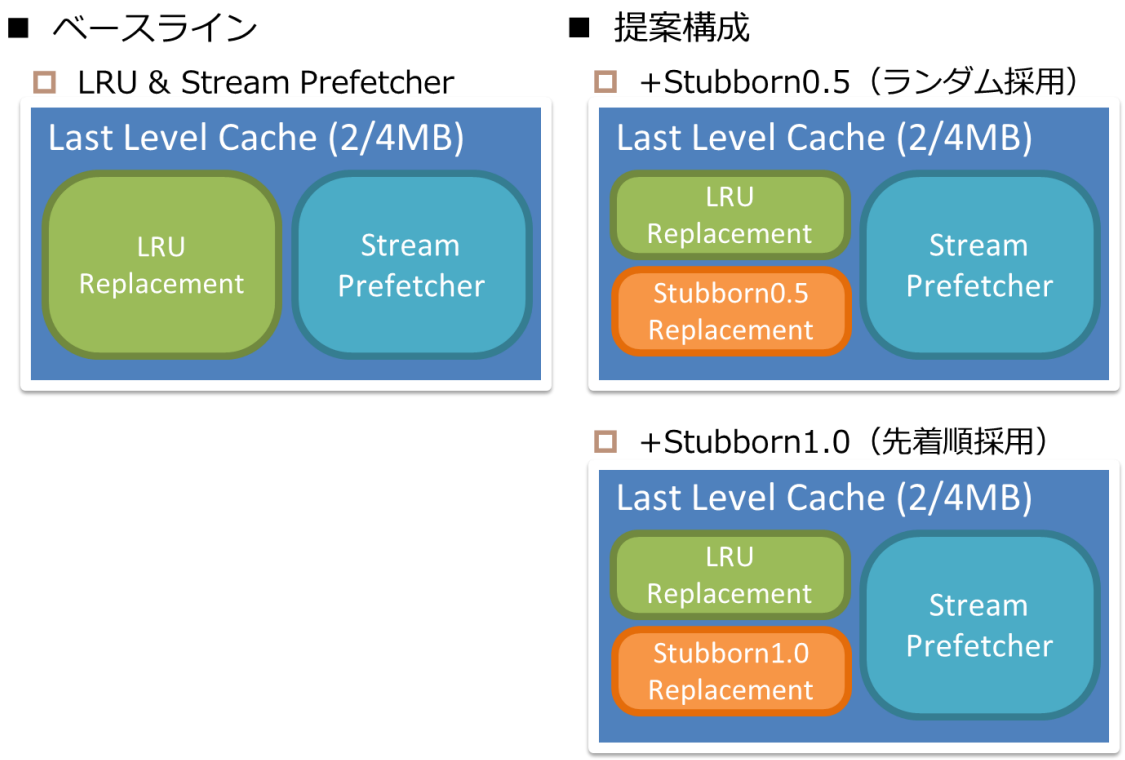
■ ベースライン
LRU & Stream Prefetcher
Last Level Cache (2/4MB)
LRU Replacement
Stream Prefetcher
■ 提案構成
+Stubborn0.5（ランダム採用）
Last Level Cache (2/4MB)
LRU Replacement
Stubborn0.5 Replacement
Stream Prefetcher
+Stubborn1.0（先着順採用）
Last Level Cache (2/4MB)
LRU Replacement
Stubborn1.0 Replacement
Stream Prefetcher

図 8.1　評価モデル

第9章 評価

9.1. 性能評価

提案手法によりキャッシュミスを削減したことで，プロセッサとして性能がどれだけ向上するかを評価するための IPC 評価と， 第 6 章でのポテンシャル評価との比較のための MPKI 評価を示す．IPC，MPKI の評価共に，8.1.1 節で述べたように，ベースラインとして使用する LRU のプリフェッチに関する動作の 2 種類をそれぞれ別に提案構成と比較する．

ここで，RIPC は，各ワークロードのベースラインでの IPC を 100%としたときの相対的な IPC を意味する．

9.1.1. IPC 評価

Prefetch aware な LRU をベースラインとしたときの RIPC の比較を，キャッシュ容量 2MB，4MB それぞれの条件で図 9.1，図 9.2 に示す．

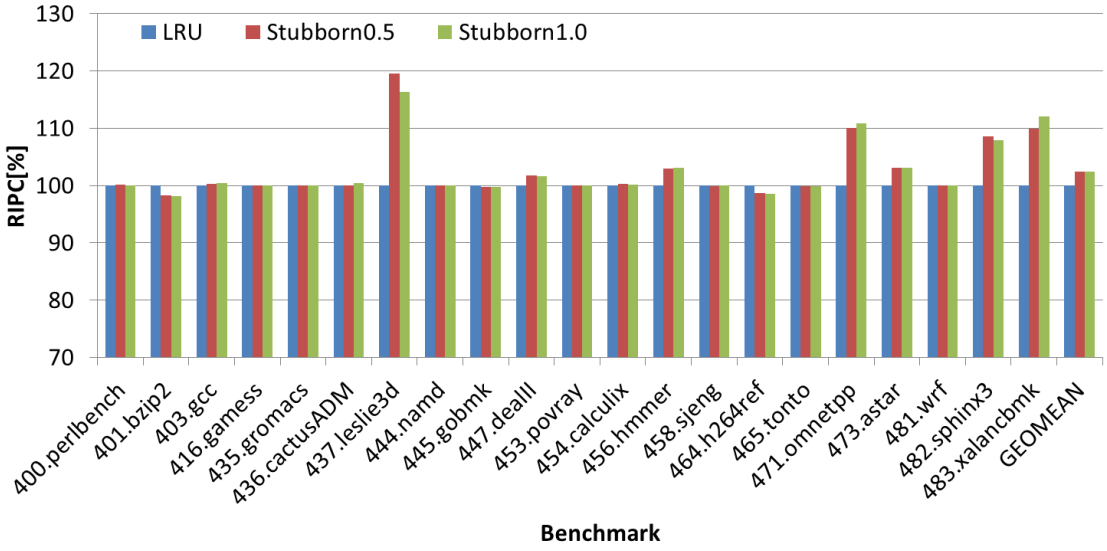

図 9.1 Prefetch aware な LRU をベースラインとする，2MB 構成での相対的 IPC

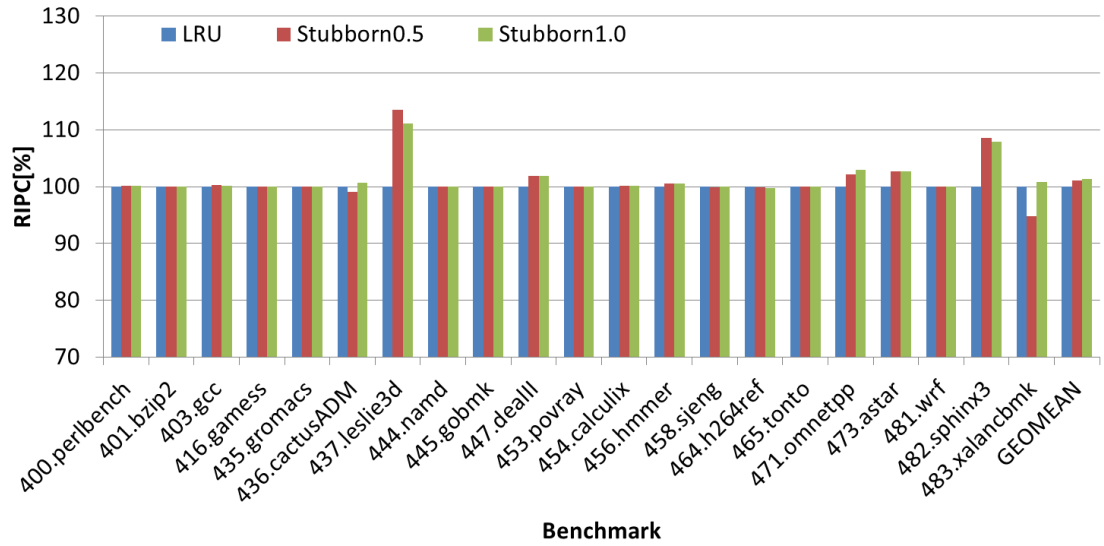

図 9.2　Prefetch aware な LRU をベースラインとする，4MB 構成での相対的 IPC

Prefetch friendly な LRU をベースラインとしたときの RIPC の比較を，キャッシュ容量 2MB，4MB それぞれの条件で図 9.3，図 9.4 に示す．

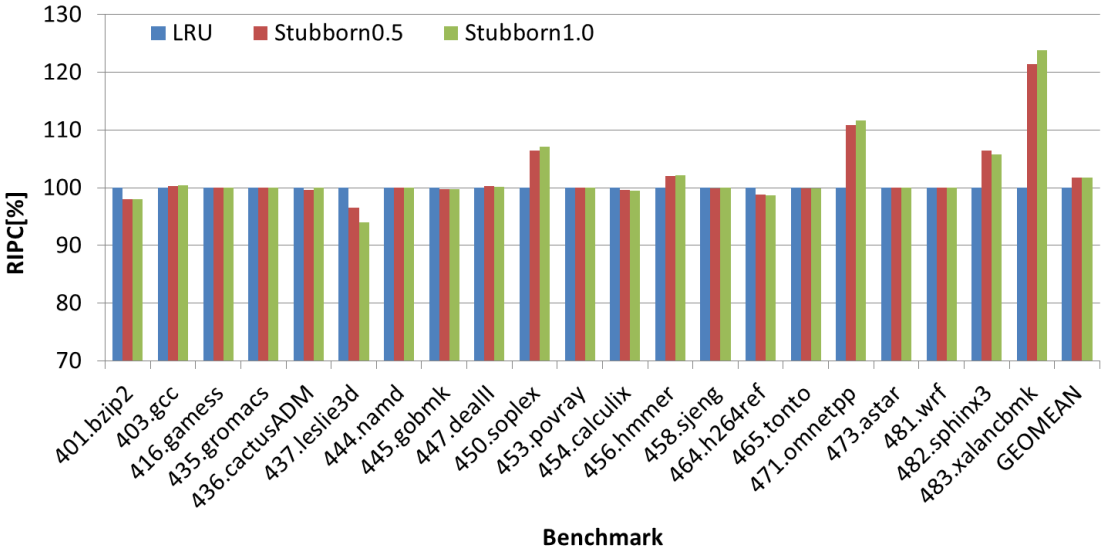

図 9.3　Prefetch friendly な LRU をベースラインとする，2MB 構成での相対的 IPC

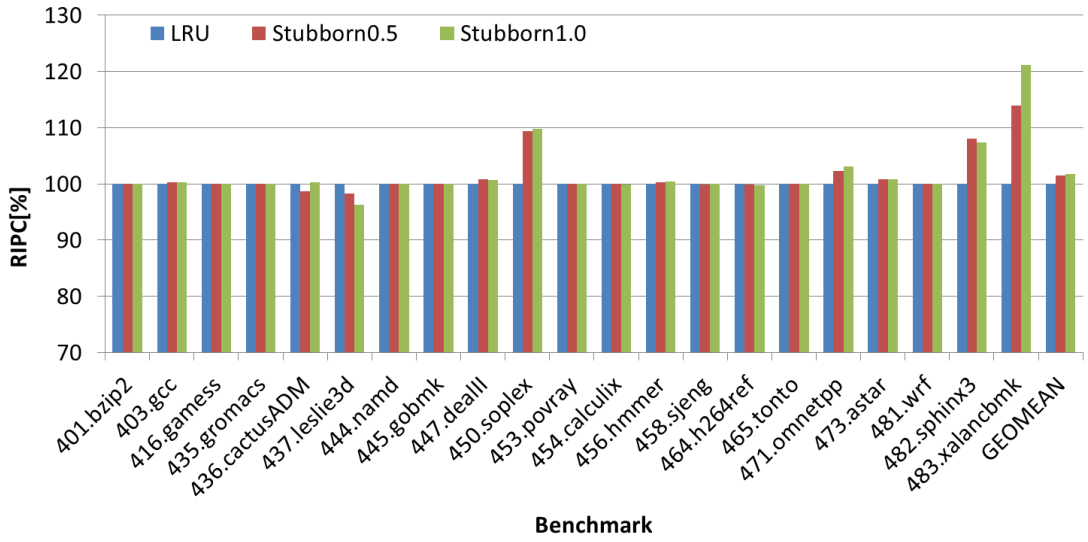

図 9.4　Prefetch friendly な LRU をベースラインとする, 4MB 構成での相対的 IPC

これらの IPC 評価で，最もベースラインからの IPC 向上が大きいのは Prefetch friendly な LRU をベースラインとしたときの Stubborn1.0 による 2MB 構成の 483.xalancbmk における 23.9%向上で, 次点は同じワークロードで Stubborn0.5 構成のときの 21.5%向上である．同ワークロードでは，4MB 構成の場合も Stubborn1.0 で 21.1%の向上と，大きな性能向上を得られた．到着順やランダムを持って確保するラインを決める手法でこれほどの性能向上が見られるのは面白い.

他に特筆すべき性能向上を得られたのは 471.omnetpp で, いずれの LRU ベースで, Stubborn0.5，1.0 いずれの選択アルゴリズムでも 10%を超える性能向上を達成している.

幾何平均でも, 2MB 構成, Prefetch aware な LRU ベースで Stubborn0.5 は 2.40%, Stubborn1.0 は 2.38%の性能向上. Prefetch friendly な LRU ベースでは Stubborn0.5 は 1.73%，Stubborn1.0 は 1.74%の伸びを見せた．4MB 構成では，Prefetch aware な LRU ベースで Stubborn0.5 は 1.05%, Stubborn1.0 は 1.33%の性能向上. Prefetch friendly な LRU ベースでは Stubborn0.5 は 1.45%，Stubborn1.0 は 1.73%の伸びを見せた．最も伸びが大きいのは，2MB 構成，Prefetch aware な LRU ベースでの Stubborn0.5 の 2.40%向上である．傾向としては 2MB であれば Prefetch aware な LRU ベースのほうが性能の伸びが大きく，4MB なら Prefetch friendly な LRU ベースの方が伸びが大きい.

性能の向上と低下の両方が見られるのが 437.leslie3d で，Prefetch aware な LRU ベースでは最大 19.6%の IPC 向上が得られたが，Prefetch friendly な LRU ベースでは最大 6.0%の性能低下が生じている．これについては後ほど考察する.

9.1.2. MPKI 評価

Prefetch friendly な LRU をベースラインとしたときの MPKI の比較を，キャッシュ容量 2MB，4MB それぞれの条件で図 9.5，図 9.6 に示す.

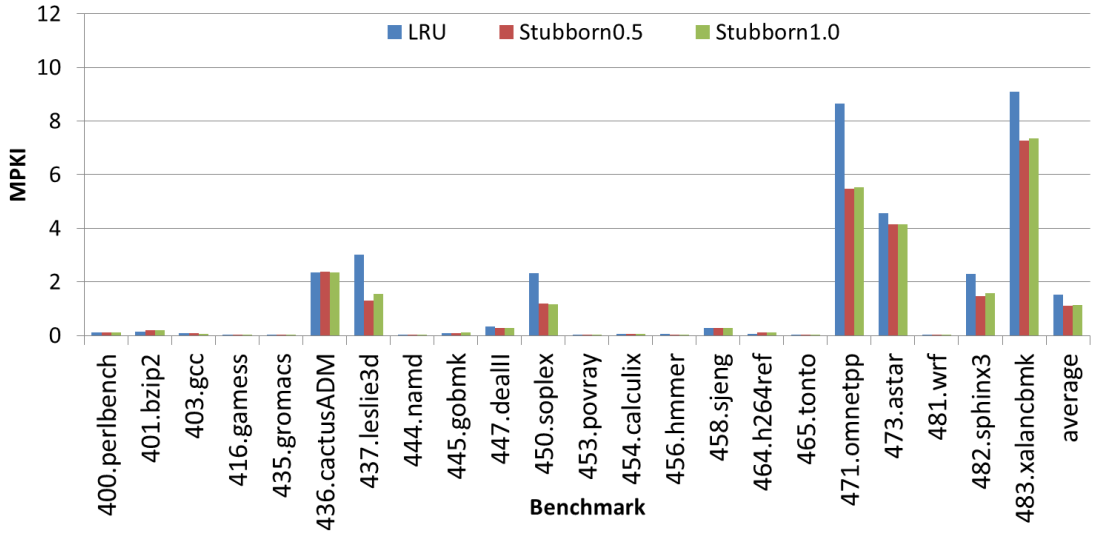

図 9.5　Prefetch aware な LRU をベースラインとする，2MB 構成での MPKI

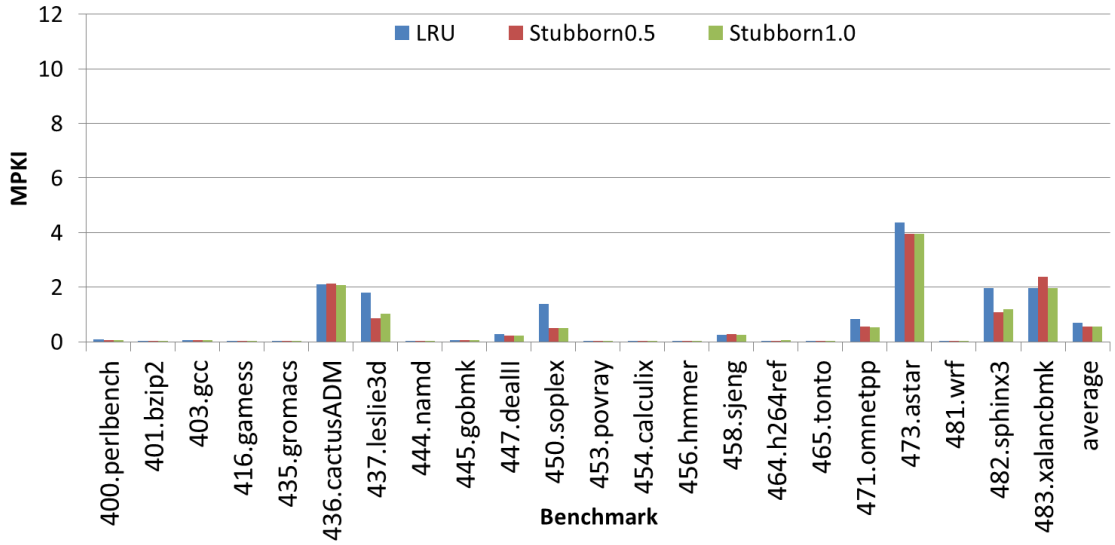

図 9.6　Prefetch aware な LRU をベースラインとする，4MB 構成での MPKI

Prefetch friendly な LRU をベースラインとしたときの MPKI の比較を，キャッシュ容量 2MB，4MB それぞれの条件で図 9.7，図 9.8 に示す.

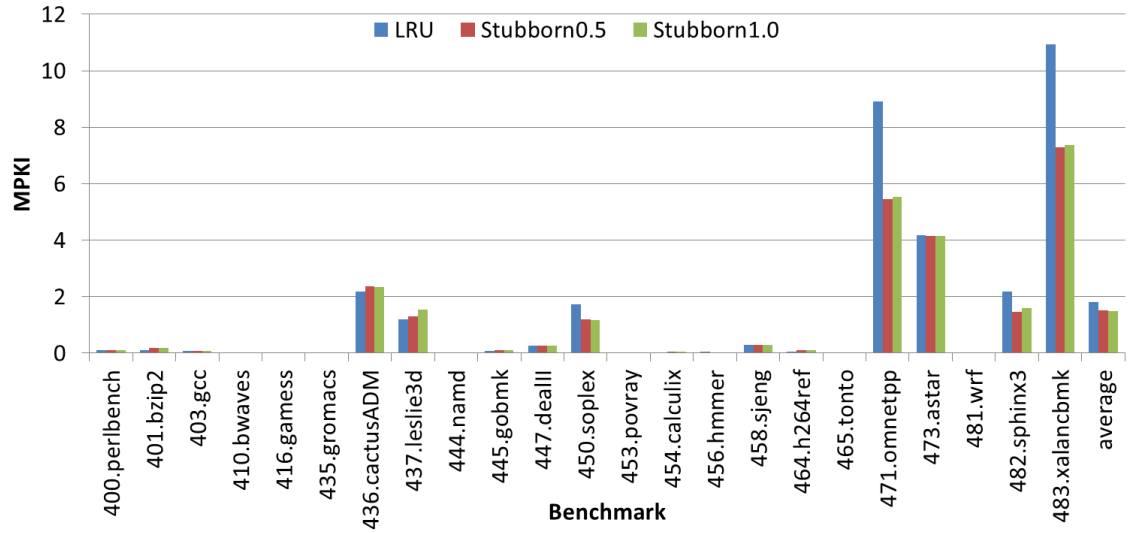

図 9.7　Prefetch friendly な LRU をベースラインとする，2MB 構成での MPKI

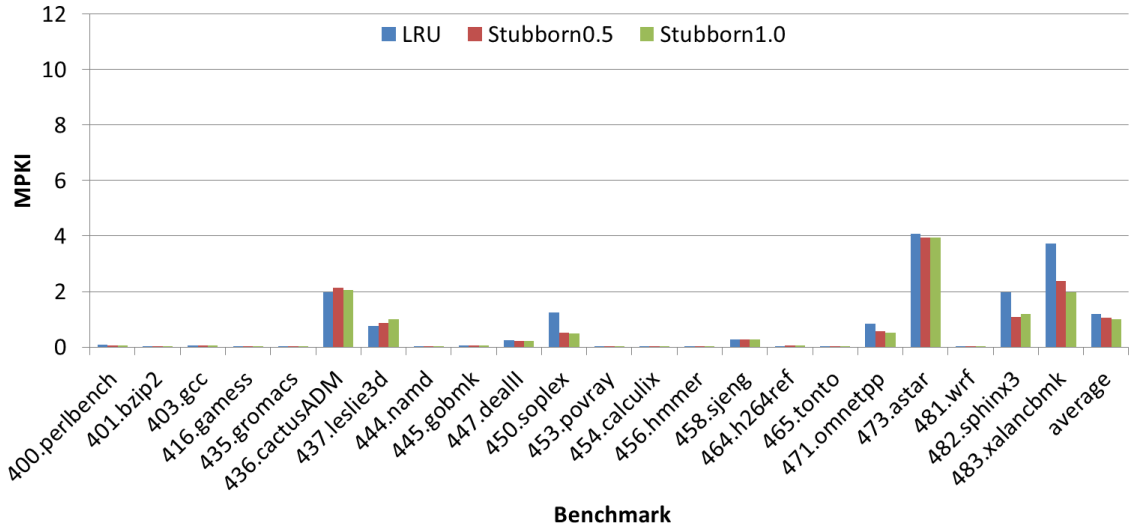

図 9.8　Prefetch friendly な LRU をベースラインとする，4MB 構成での MPKI

これらの MPKI 評価から，IPC が伸びている構成ではミス数の減少率も大きいことが見て取れる．483.xalancbmk や 471.omnetpp といった，元の MPKI が大きいワークロードで顕著となる．そのほか，絶対的には MPKI は小さい 450.soplex でも，そのミス削減率は IPC に効果として強く現われている．

ミス数が悪化している例では，Prefetch friendly な LRU ベースでの 436.cactusADM や 437.leslie3d がある．これらは確かに IPC でも性能を落としている．

これらの MPKI を通して見ても, 第 6 章における図 6.2 の評価で使用した共通のワークロードでは同様の傾向が見られている．第 6 章での評価条件と違って理想的なキャッシュライン選択ができているわけでないにも関わらず, 同様の MPKI 削減を達成できているのは, LLC の大容量をもってして, 第 6 章での評価に使用した 64KB のテーブルより遥かに大きな 1MB，2MB の領域を Stubborn 領域としてつぎ込んでいるからである．

9.2. 考察

評価結果から，Stubborn 戦略がどのように働いたかを議論する．特にワークロードと Stubborn 戦略，性能の評価結果の関係について議論する．

9.2.1. ワークロードとの相性

Stubborn 戦略と各ワークロードの相性について述べる．図 9.9 に Prefetch aware な LRU をベースラインとしたときの, ワークロードごとのアルゴリズムと IPC の変化を示す．多くのワークロードでは，いずれのアルゴリズムであっても大差が無く横一列に並んでいる．これらのワークロードは，512KB の L2 キャッシュでほとんどのやりくりが済んでいるため L3 の容量の恩恵を受けておらず, それゆえ L3 に適用されるキャッシュアルゴリズムが変わっても変化がない．

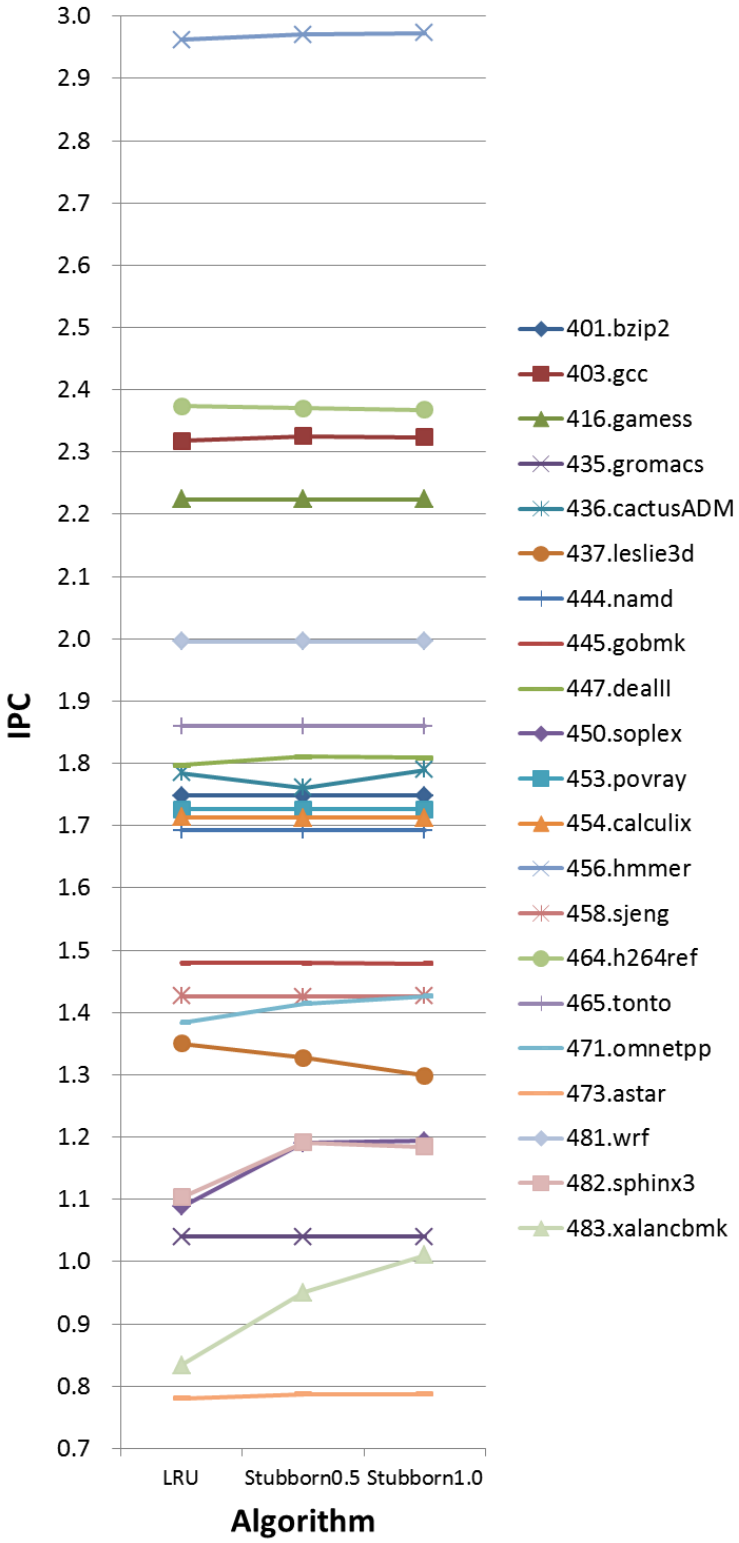

図 9.9　ワークロードごとのアルゴリズムと IPC の変化

性能が伸びているキャッシュアクセスパターンを持つワークロードはどのようなものかというと，ストリームアクセスが使用するアドレスの合計容量が数 MB オーダーの範囲でギリギリ L3 の容量より大きく，一度のストリームのアクセスが数 M 命令の周期で一周するようなアクセスパターンを持つワークロードである．具体的には 471.omnetpp（図 9.10）や 483.xalancbml（図 9.11）がこれにあたる．（図のアクセスパターンは共にベースラインのもの）

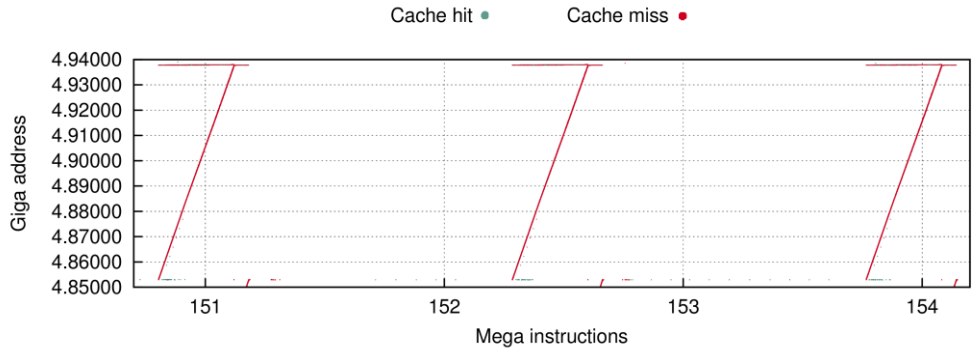

図 9.10　471.omnetpp のアクセスパターン

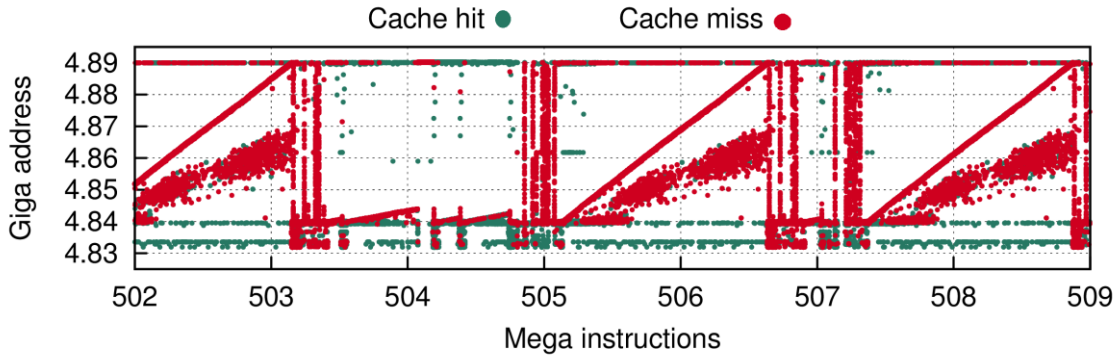

図 9.11　483.xalancbml のアクセスパターン

これに対して，L3 を利用するワークロードでも MPKI が減少せず，IPC の伸びがないワークロードがある. 特に, 使用アドレス数が数十 MB オーダーと多く長く伸び，一度のストリームアクセスも時間的に 100M オーダーの命令数を掛ける広い範囲のストリーミングアクセスを中心としたアクセスパターンを持つワークロード，例えば 473.astar のようなワークロード（図 9.12）とは相性が悪く，性能を伸ばすことができない．一方で，このようなワークロードは L3 の大容量を Stubborn 戦略に半分持って行かれることがなかったとしても，再参照までに有効活用することができないので，Stubborn 戦略を適用しようとしまいと大して性能が変わらないという結果になる．

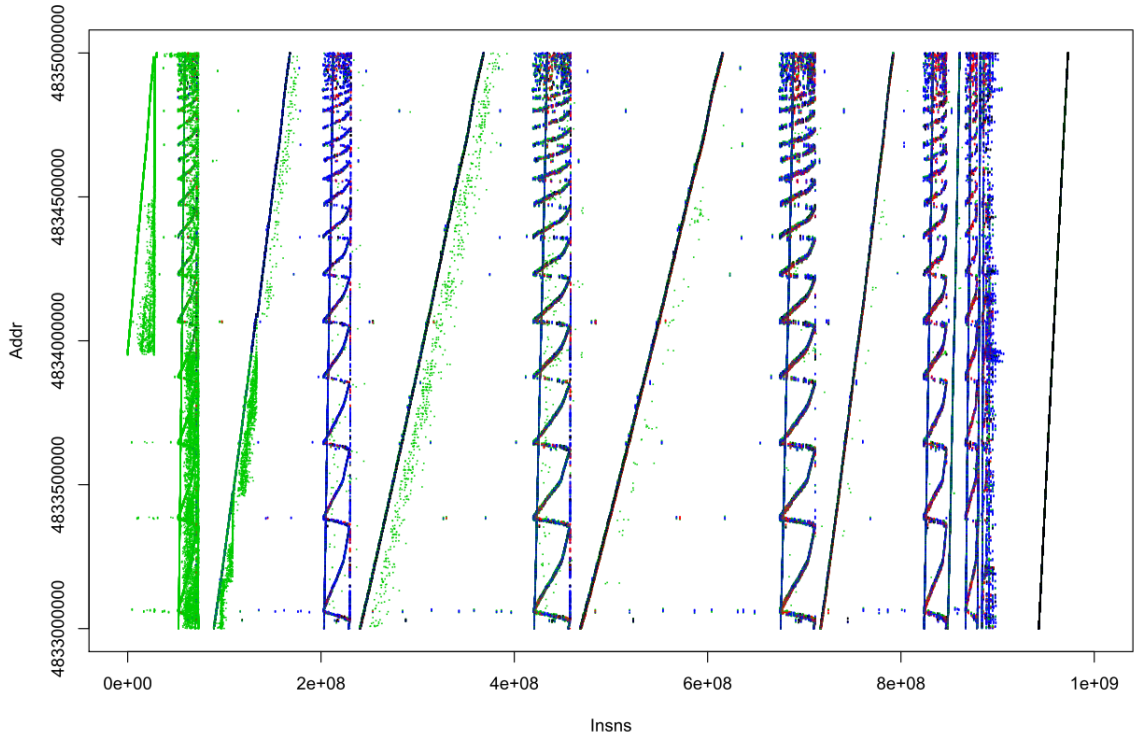

図 9.12　473.aster のアクセスパターン

437.leslie3d について, Prefetch aware な LRU での評価に対して, Prefetch friendly な LRU での評価では，RRIP，Stubborn 共に相対的な IPC の伸びが無く，むしろ下がっている．これは，437.leslie3d のワークロードの特長としてプリフェッチが良く効くワークロードであるため，ベースラインの IPC が高くなったことで，それ以外の IPC が相対的に落ちたである．より具体的には，プリフェッチしてきたものを LRU オーダーで MRU に置いてしまうことが直接性能に寄与するためである．この IPC の低下は Stubborn で可換なライン数が減少した場合だけでなく，RRIP でプリフェッチしたものを直接 MRU に置けなくなっている場合にも同様に発生していることから，その特性がはっきりと表れていることがわかる．

9.2.2.　選択アルゴリズムによる違い

今回の評価で実装，評価した追い出し不可属性付与対象選択アルゴリズムによる差について議論する．

先着順採用は，今回の評価ではいずれもベースラインに勝る性能を示している．先着順採用という単純な方式でキャッシュミスを救えている理由は，その LLC の大容量性と，比較的大きめなウェイ数を活かせている点である．もちろん，先着で来たアクセスの中にはその後一切アクセスがこないデッドブロックとなるラインもあるが，

そのようなラインを拾う可能性があったとしても，それ以上に LLC の大容量と 8 ウェイという大きなウェイ数の恩恵が大きかった．

また，先着順採用とはいえど，今回の評価ではワークロードの頭 10G 命令はスキップした後の 1G 命令を評価対象としていて，それゆえにほとんどのワークロードではその走り始めでの初期化動作のためのたった一度しか利用されないようなコード，データを抱え込んでしまうことが避けられている．初期化的な動作を終えた後の主たる実行フェーズにいることで，繰り返しアクセスの多いフェーズで対象を選択することができている．

対して，ランダム採用の選択アルゴリズムでは，ほとんどのワークロードで，IPC の伸びとキャッシュミスの削減効果が先着順採用に劣っている．これは，キャッシュにきたラインを必ずしも追い出し不可属性に採択しなかったことで，Stubborn 領域の使用率が低下し，Stubborn キャッシュとして機能し始めるまでが遅れたためにその効果が薄れたパターンである．

第10章 今後の展望

10.1. キャッシュライン選択手法

Stubborn 戦略に基づくキャッシュシステムを実プロセッサ上に実現するためには，より良いキャッシュラインを選択する手法，アルゴリズムを確立する必要がある．

先着順，ランダム以外での実現可能性について考える．実行履歴の利用．また，5.2.3.3 節において述べた，追い出し不可属性の付加の判断を追い出し時まで遅延させられるという特長を生かした選択手法についても，今回の研究では実現させていない．今後は，このキャッシュライン挿入から追い出しまでの期間にモニタリングを行い，追い出し不可属性の付加の判断材料にすることで，第 6 章で評価したような，Stubborn 戦略における理想的なキャッシュライン選択へさらに近づける．

また，先行研究として，スマートなキャッシュアルゴリズム提案される以前から存在する手法としてキャッシュロック[10]がある．これは，キャッシュの一部領域，もしくは専用の高速なメモリ領域に，プログラム上で予め指定したアドレス領域のデータを格納し続ける手法である．これを実現し効果を引き出すためには，プログラミングにおいてどのデータをキャッシュロックするかを見定める，熟練の人間による判断を必要とする．この機構は SPARC, ARM をはじめとするいくつかの商用プロセッサに搭載されている．この手法は，キャッシュに永続的に残し続ける変数を予め定め，保持し続けるという点では，Stubborn 戦略と共通性を持つ．しかしながら，キャッシュロックの場合，専用のキャッシュ領域に保持するデータまたはプログラムにあたるアドレス領域を，プログラムの記述時からあらかじめ静的に，かつプログラムチューニングに熟練した人間の判断に基づいて決定しなければならず，また実行時の履歴を利用したキャッシュラインの選択ができないという問題がある．

10.2. 他のキャッシュ構成要素との協調

さらに，本研究の過程において，Stubborn 戦略に基づくキャッシュアルゴリズムは，以下の各節に挙げるキャッシュ制御のテクニックと共存，協調した動作をすることができると推測している．これらの機構との協調により，Stubborn 戦略により得られるキャッシュミス削減，性能向上，あるいは電力削減の効果をさらに大きくすることを狙う．

10.2.1. ベースとする置き換えアルゴリズムの試行

本論文の評価では，LRU をベースとして Stubborn 戦略を実装した．評価でもって RRIP を比較対象の 1 つとしたが, Stubborn 戦略と RRIP はその実現について排他ではなく，RRIP をベースとした Stubborn 戦略実装のキャッシュも実現できる．その場合，相乗効果により性能がさらに底上げできることが期待される．

10.2.2. アダプティブな機構化

キャッシュアルゴリズムの適用についてのテクニックに，アダプティブ機構化[21]がある．アダプティブというのは，2 つ以上の手法があるときに，その時々で適した方の手法を適用することである．関連研究にて紹介した DRRIP についても，これは元の置き換えアルゴリズムである RRIP をアダプティブにしたものである．

今回の Stubborn 戦略をアダプティブとすることで，例えば Prefetch friendly な LRU をベースラインとする場合の 437.leslie3d のような，Stubborn 戦略を適用することで悪影響を出す場面ではそれを無効にして実行させることで，悪影響を防ぐことができるようになる．

10.2.3. Stubborn 領域サイズの動的な変更

今回の評価では，Stubborn 戦略が 1 セットに占める領域（ウェイ数）の割合を一律 50%，ウェイ数が 8 ウェイのセットアソシアティブテーブルのキャッシュであれば 4 ウェイまで Stubborn 領域として使用した．この領域の割合について，実行時に動的に変動させることで，ワークロードやフェーズごとに最適な Stubborn 領域の割合に遷移させる機構が考えられる．この機構は，パーティショニング[20,21]として知られる手法により実現できる．

また，この適用割合をゼロにすることで，10.2.2 節で先に述べたアダプティブな機構化として，Stubborn 戦略を適用しないキャッシュとの切り替えも実現できる．

10.2.4. 半導体リソースの増大と Stubborn 戦略

トランジスタ数増加の傾向は，半導体集積度の向上に伴う性能向上を示すムーア則に従って，今後も継続すると見られている[14,19]． 本節ではこの傾向を，Stubborn 戦略を活かすための手段として考えてみる．

10.2.4.1. 多ウェイ数キャッシュテーブル

Stubborn 戦略を単一のキャッシュテーブルに実装する場合，1 つのセットを Stubborn 領域とベースアルゴリズムで管理される領域で分かち合うことになる．つまり，Stubborn 領域が大きくなった後は，ワークロードによっては競合性ミスが起きるリスクが高まる．そこで，1 つのセットがもつウェイ数を大きくするという対策が考えられる．

半導体集積度の向上に伴い，キャシュメモリ容量や 1 セットあたりのウェイ数も増加してきた．実際に，Intel の商用プロセッサ StorngARM（SA-1100）[12,13]では，32 ウェイのセットアソシアティブという，1 セットあたりのウェイ数が非常に大きい構成が採用されているプロセッサがある．また，このようにウェイ数の大きいセットを持つキャッシュテーブルを効率的に利用する方式についての研究も存在する[11]．このプロセッサのキャッシュメモリのように 32 ウェイもあれば，そのうち 4 分の 1 や半分ぐらいのウェイを Stubborn 領域で占領しても，まだ競合性ミスの誘発は避けやすい．一方で，1 セットの中に，時間的局所性に特化した FIFO 管理のウェイ再参照に特化した Stubborn 戦略で管理されるウェイが混在するのは相性が良く，管理のためのハードウェアコストとしては安く実現できる．

10.2.4.2. 別テーブルで実現する Stubborn キャッシュ

10.2.4 節で，ハードウェアリソースとしてのトランジスタ数増加とキャッシュ容量増加について述べた．しかしながら，キャッシュメモリの場合は，そのレイテンシを抑えたいという要求から，一定以上のサイズのセット数，ウェイ数で実現することは難しくなる．そのような状況において，増え続けるトランジスタ数のリソースをつぎ込む先の 1 つとして Stubborn 戦略だけで構成される，LLC とは別のテーブルで実現される Stubborn キャッシュが考えられる．

Stubborn キャッシュ専用のテーブルでは，タグの更新が不要となるので，読み書きの機構が簡略化でき，レイテンシを小さくできるのではないかと考える．電力的にも通常のキャッシュテーブルより優位に構成できる．ただし，この方策については，複数のキャッシュに問い合わせ，その結果をすり合わせるための追加機構のオーバヘッドと電力の増分を気にする必要がある．

10.2.4.3. 不揮発メモリによるキャッシュ

不揮発メモリを合わせたヘテロなキャッシュメモリの構成が，Stubborn 戦略に利するのではないかと目論む．

キャッシュメモリを構成するデバイスを，SRAM から不揮発メモリに転換する試みがある．そのメリットは，面積当たりの容量増加と電力削減にある．これは最近のキャッシュメモリを構成するデバイスに関する研究のトレンドとなっている．

この不揮発メモリは，Read については SRAM と同程度のレイテンシで実現できるが，Write は SRAM の数倍から十数倍の時間を要する．この特性は，Stubborn 戦略と相性がよい．なぜなら，Stubborn 戦略では 10.2.4.2 節でも述べたように，タグと置き換えオーダー管理ビットの更新が不要となるからである．これによって，Write が遅いというデメリットの影響を抑えたまま，不揮発メモリによる低消費電力性，大容量性が得られる．

第11章 おわりに

プロセッサアーキテクチャにおいて，メモリサブシステム，キャッシュシステムの性能は依然として主要な課題の一つである．近年では半導体微細化に伴いキャッシュメモリ，特にラストレベルキャッシュ（Last Level Cache, LLC）の大容量化が進み，これを上手く利用するためにプリフェッチに協調するスマートなキャッシュ置き換えアルゴリズムが提案されている．

しかしながら，これらの賢いキャッシュアルゴリズムをもってしても救えていないキャッシュミスが存在する．本研究の調査で，それは一度アクセスされたあるアドレスに対応するラインがキャッシュに載った後，実行命令数が 1M 命令以上経過した後に再度アクセスされた場合に生じるキャッシュミスであることが明らかになった．本研究ではこれを長期再参照ミスと命名した．これらの調査から，特に LLC でのキャッシュミスのボリュームゾーンは長期再参照ミスであることがわかった．

この調査を踏まえて，本研究では，ある基準によって定めたキャッシュラインを長期間追い出さないことを特長としたキャッシュアルゴリズムの基本戦略を提案する．実行履歴に基づき置き換えを起こすアルゴリズムによって救えないキャッシュミスであるならば，いっそヒストリに頼った直近の再参照に期待するのはやめて，置き換えを起こさない手法として試行した．我々はこれに Stubborn 戦略と名付けた．“Stubborn”は“頑固”を意味する． Stubborn 戦略は，キャッシュ中に追い出しを起こさない領域をつくることで実現させる．

本提案手法を，LRU をベースに実装し，シミュレーションによる評価を行った．このようなシンプルな戦略にも関わらず，結果として最大で 23.9%の IPC 向上，幾何平均でも 2MB 構成，Prefetch aware な LRU ベースでのランダム採択を行う Stubborn キャッシュで最大の 2.40%向上が得られた．

謝辞

まずはこのような破天荒とさえ言える本研究の提案にまでゴーサインを下さり，絶えず丁寧な御指導を賜りました，東京大学准教授 入江英嗣先生に深く感謝致します．本研究について，多くの助言や議論の機会をいただき，特に計算機資源について多大なる協力をいただきました吉永努教授，吉見真聡助教，中島拓真先輩，城間隆行君，川原尚人君に深謝いたしております．また，特別研究学生として坂井・入江研究室に受け入れていただきました，東京大学教授 坂井修一先生と八木原晴水様，同研究室の皆様に感謝いたします．

本研究は，科研費若手研究 25730028「新アーキテクチャによる高効率プロセッサコアおよびそのマルチコア構成の研究」の助成を受けたものであり，科研費による学術への支援に感謝いたします．また，株式会社中山鉄工所様に多大なるご支援を賜りました．感謝いたします．

参考文献

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

[2] Standard Performance Evaluation Corporation: SPEC CPU 2006 Benchmark Suite, http://www.spec.org/cpu2006/.

[3] A. Jaleel, K. B. Theobald, S. C. Steely,Jr., and J. Emer, “High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP),” Proceedings of the 37th Annual International Symposium on Computer Architecture, New York, NY, USA, pp.60–71, 2010.

[4] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, “Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers,” IEEE 13th International Symposium on High Performance Computer Architecture, 2007. HPCA 2007, pp.63–74, Feb. 2007.

[5] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the Pentium 4 processor. Intel Technology Journal Q1 2001 Issue, Feb. 2001.

[6] J. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. IBM Technical White Paper, Oct. 2001.

[7] K. J. Nesbit and J. E. Smith, “Data Cache Prefetching Using a Global History Buffer,” Proceedings of the 10th International Symposium on High Performance Computer Architecture, Washington, DC, USA, pp.96, Feb. 2004.

[8] N. Anchev, M. Gusev, S. Ristov, and B. Atanasovski, “Intel vs AMD: Matrix Multiplication Performance,” 36th International Convention on Information Communication Technology Electronics Microelectronics (MIPRO), pp.182–187, May 2013.

[9] N. P. Jouppi, “Improving Direct-mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers,” 17th Annual International Symposium on Computer Architecture, 1990. Proceedings, pp.364–373, May 1990.

[10] Y. Liang, T. Mitra, and L. Ju, “Instruction Cache Locking Using Temporal Reuse Profile,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.34, no. 9, pp.1387–1400, Sep. 2015.

[11] D. H. Albonesi, “Selective cache ways: on-demand cache resource allocation,” 32nd Annual International Symposium on Microarchitecture, 1999. MICRO-32, pp.248–259, 1999.

[12] Intel, “Intel StrongARM SA-1100 Microprocessor for Embedded Applications,” Jun. 1999.

[13] Intel, “Intel XScale Core Developer’s Manual,” Jan. 2004.

[14] M. Stettler and S. Krishnapura,
“Moore’s Law – Not Dead – and Intel’s Use of HPC to Keep it Alive,” http://www.hpcwire.com/2016/01/11/moores-law-not-dead-and-intels-use-of-hpc-to-keep-it-that-way/, Jan. 11. 2016.

[15] D. Sanchez and C. Kozyrakis, “Vantage: Scalable and Efficient Fine-grain Cache Partitioning,” Proceedings of the 38th Annual International Symposium on Computer Architecture, New York, NY, USA, pp.57–68, 2011.

[16] Hassan Ghasemzadeh, Sepideh Sepideh Mazrouee, and Mohammad Reza Kakoee, “Modified Pseudo LRU Replacement Algorithm,” In ECBS, pp. 368–376. IEEE Computer Society, 2006.

[17] C.-J. Wu, A. Jaleel, M. Martonosi, S. C. Steely Jr., and J. Emer, “PACMan: Prefetch-aware Cache Management for High Performance Caching,” Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, pp.442–453, 2011.

[18] 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, pp. 120-121, May. 2009.

[19] S. M. Cea, S. Botelho, A. Chaudhry, P. Fleischmann, M. D. Giles, A. Grigoriev, A. Kaushik, P. H. Keys, H. W. Kennel, A. D. Lilak, R. Mehandru, M. Stettler, B. Voinov, N. Voynich, C. Weber, and N. Zhavoronok, “Process Modeling for Advanced Device Technologies,” J Comput Electron, vol.13, no. 1, pp.18–32, Aug. 2013.

[20] M. K. Qureshi and Y. N. Patt, “Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches,” 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006. MICRO-39, pp.423–432, Dec. 2006.

[21] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely Jr., and J. Emer, “Adaptive Insertion Policies for Managing Shared Caches,” Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, New York, NY, USA, pp.208–219, 2008.

[22] H. Irie, T. Miyoshi, G. Honjo, K. Hiraki, and T. Yoshinaga, “Using Cacheline Reuse Characteristics for Prefetcher Throttling,” IEICE TRANSACTIONS on Information and Systems, vol.E95-D, no. 12, pp.2928–2938, Dec. 2012.

[23] Compaq Computer Corporation, “Alpha Architecture Handbook Version 4,” Oct. 1998.

[24] Bruce Hutton, “The Alpha Computer Architecture,” Jan. 2007.

[25] 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,” 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp.265–274, May 2011.

発表論文

本研究に関して得られた成果について，次の 2 件の論文発表を行った．

(1) Hayato Nomura, Takuma Nakajima, Masato Yoshimi, Tsutomu Yoshinaga, Hidetsugu Irie, “Stubborn Cache: A Novel Strategy for Repeating Thrashing Access Patterns,” The 18th International Symposium on Low-Power and High-Speed Chips (COOL Chips XVIII) , pp. 19, Yokohama, Japan, Apr. 2015.
（査読あり国際会議，ポスター発表，**Featured Poster Award 受賞**）

(2) 野村隼人, 力翠湖, 吉見真聡, 吉永努, 入江英嗣, “動的推定によるキャッシュパーティショニング最適化,” 情報処理学会研究報告, Vol. 2014-ARC-211, No. 2, pp. 1-6, 新潟, 2014 年 7 月．（査読なし研究会，口頭発表）

本研究の中核となる Stubborn 戦略の提案とポテンシャル評価について，査読付き国際会議発表 1 件の発表を行った(1)．この国際会議 COOL Chips では Featured Poster Award を受賞した．

また，発表(2)の研究会発表 1 件は，本研究に先行して行った，キャッシュメモリにおけるパーティショニングに関する研究である．1M 命令を超える長期間で再参照されるアドレスで発生するキャッシュミスに着目するきっかけとなった．