マルチコア・プロセッサの不均質共有キャッシュにおけるLRU大域置き換えアルゴリズム

塩谷亮太[†] ルォンディンフォン[†] 入江英嗣[††]
五島正裕[†] 坂井修一[†]

本稿では，マルチコア・プロセッサの不均質共有キャッシュ（non-uniform shared cache）における，LRU に基づいた大域置き換えアルゴリズムを提案する．マルチコア・プロセッサでは複数のプロセッサ・コアが同一のチップ上に存在していることから，プロセッサ間の緊密な通信環境が構築可能である．したがって，各プロセッサ・コアが独立して持つキャッシュはお互いに高速な参照が可能であり，これを利用してキャッシュの共有を行う構成を不均質共有キャッシュと呼ぶ．本稿ではこの不均質共有キャッシュにおいて，単一の LRU 情報に基づき，キャッシュ・ラインの置き換えを行う手法について述べる．本手法では，使用されていないキャッシュ・ラインを他のコアの高次キャッシュとして用いることが可能であり，従来の共有キャッシュ構成と比較して平均で 11%の性能向上を達成できた．

LRU-based Global Replacement Algorithm for Non-uniform Shared Cache of Multi-core Processors

Ryota Shioya,[†] Luong Dinh Hung,[†] Hidetsugu Irie,[††]
Masahiro Goshima[†] and Shuichi Sakai[†]

In this paper, we propose LRU-based global replacement algorithm for Non-Uniform Shared Cache of multi-core processors. Multi-core processor allows fast communication between processors because the cores are in the same chip. Therefore, the caches, independently placed in each core, can be referenced by each other with low latency. We call such a cache sharing structure Non-Uniform Shared Cache. We describe a replacement algorithm of cache-lines based on a global LRU information in Non-Uniform Shared Cache. This algorithm can use idle line as higher-level cache for the other. Our technique archived performance improvement compared with conventional shared cache by 11%.

1. はじめに

近年のプロセス技術の向上により，単一のチップ上に複数のプロセッサ・コアを集積する**マルチコア・プロセッサ**が商用化された．マルチコア・プロセッサでは，チップ外へのインタフェースを各コア間で共有するため，コア 1 つあたりのメモリ・バンド幅が相対的に縮小する．そのためマルチコア・プロセッサでは，チップ外へのアクセスをできる限り減らす必要があり，シングルコア・プロセッサに比べ，オンチップ・キャッシュのヒット率がいっそう重要となる．同時に，マルチコア・プロセッサでは，コア間の通信バンド幅に対する制約が緩和されるため，チップ内キャッシュの構成を工夫する余地が広がる．したがって，マルチコア・プロセッサでは，オンチップ・キャッシュの構成が重要な差別化要因になると考えられる．

マルチコア・プロセッサのキャッシュ構成

マルチコア・プロセッサのキャッシュ構成は，大きく **private** と **uniform shared** に分けられる．private 構成と uniform shared 構成は，容量効率とアクセス・レイテンシについてそれぞれ相反する性質を持っている．

各コアが独立したキャッシュを持つ private 構成では，それぞれのキャッシュが同一のラインを持ちうるため，容量効率が低くなる傾向にある．一方，各コアで単一のキャッシュを共有する uniform shared 構成では，同一ラインのコピーはたかだか 1 つしか存在しないため，データ・アレイの利用効率が向上する．しかし，uniform shared 構成では，コアとキャッシュの

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 科学技術振興機構
Japan Science and Technology Agency

間に相互結合網が入るため，コアからのアクセス・レイテンシが増大してしまうという問題がある．

non-uniform shared cache

private 構成，uniform shared 構成に対し，低レイテンシと高容量効率の両立を狙ったものとして，**non-uniform shared cache**（不均質共有キャッシュ）がある．non-uniform shared cache は，物理的なデータ・パスの構成は private と同様としてアクセス・レイテンシを短縮しながら，キャッシュ間でラインをやりとりすることによって容量効率を向上させる手法である．容量効率の向上は，基本的には，1 度メモリからフェッチしたラインを，可能な限りチップ内に残すことによって達成される．たとえば，Terasawa ら[1] は，Z-Cache と呼ばれる手法を提案している．Z-Cache では，他のコアに存在する invalid なラインを victim cache として利用する．すなわち，他のコアに invalid なライン・フレームがあれば，対象のラインをリプレースするのではなく，invalid なライン・フレームにマイグレートするのである．マイグレート後にそのラインにアクセスがあった場合，チップ外のメモリからではなく，先ほどマイグレートした先のキャッシュからフェッチすることができる．

既存手法の問題点と提案手法

non-uniform shared cache に対しては，そのほかにも様々な提案が行われてきた．その提案の多くは，invalid や shared，exclusive といった，コヒーレンス・プロトコルにおけるラインの状態を基準として，キャッシュ間におけるラインのマイグレートを行っている．

しかし，ラインの状態を基準としてマイグレートを行う手法にはいくつか問題点がある．不適切なライン・フレームをマイグレート先として選んでしまう場合や，そもそもマイグレート先となりうるライン・フレームが存在しない場合があるのである．たとえば，各コアで独立したプロセスが稼動している状況を考える．このとき，キャッシュ上のほとんどのラインは exclusive 状態である．したがって，invalid なラインをマイグレート先として利用する手法では，ほとんど性能の向上が期待できない．

そこで我々は，ラインの状態ではなく，ラインの大域的な LRU 情報を利用してマイグレートを行うことを提案する．すなわち，**大域 LRU** によってアクセス頻度が低いと判断されたラインをマイグレート先として選択するのである．

以下，2 章では背景となるマルチコア・プロセッサにおけるキャッシュの構成について述べ，3 章では上記のライン状態を基準とする手法の問題点についてより詳しく述べる．4 章では LRU を用いた大域置き換えについて述べる．5 章では提案手法についてシミュレーションを行い，その評価を示す．6 章では関連研究について述べ，7 章でまとめる．

2. マルチコア・プロセッサのキャッシュ構成

マルチコア・プロセッサのキャッシュにおいて，その構成が問題となるのは L2 以上の高次キャッシュである．L1 キャッシュはきわめて頻繁にアクセスを受けるため，アクセス・レイテンシは極力短く保つ必要がある．したがって，通常各コアはそれぞれ独立した L1 キャッシュを持つ．

L1 キャッシュと比較して，L2 キャッシュはアクセス頻度が 1 桁以上低い．したがって，L2 キャッシュにおけるアクセス・レイテンシの重要性は L1 キャッシュに比べると小さく，アクセス・レイテンシを多少犠牲にしてでも容量効率を上げることにより，総合的な性能を向上させることができる．したがって，以降ではこの L2 キャッシュの構成について述べる．以下，特に断りなくキャッシュとした場合は L2 キャッシュのことをさすものとする．また，本稿では L2 キャッシュに対し議論を行うが，L3 や L4 等のチップ内のより高次のキャッシュとした場合でも一般性を失わない．

inclusive 構成と exclusive 構成

階層型のキャッシュ構成では，低次キャッシュの持つラインが高次キャッシュに必ず含まれる **inclusive 構成**と，低次キャッシュと高次キャッシュの間でラインを排他的に持つ **exclusive 構成**が存在する．

exclusive 構成をとることの利点は容量効率の向上にある．これは，キャッシュ階層間で排他的にラインを保持することにより，低次キャッシュの分だけ使用できる高次キャッシュの容量が増加するためである．しかし，現在の一般的なプロセッサの場合，L2 キャッシュが数 MB 程度の容量を持つのに対し，L1 キャッシュの容量は数十 KB 程度であり，L2 キャッシュと比較して非常に小さい．このため，L1 キャッシュと L2 キャッシュを exclusive 構成とした場合でも，期待できる性能向上はほとんどないといえる．

また，exclusive 構成の欠点はキャッシュ・コヒーレンス・プロトコルが複雑化することである．たとえばラインの無効化要求が行われた場合，inclusive 構成では，高次キャッシュにヒットした場合にのみ，低次キャッシュに無効化要求を行えばよい．しかし，exclusive 構成の場合，無効化要求をつねに低次キャッシュに対しても行う必要があり，オーバヘッドとなる．これを避けるために，低次キャッシュのタグを高次キャッシュ

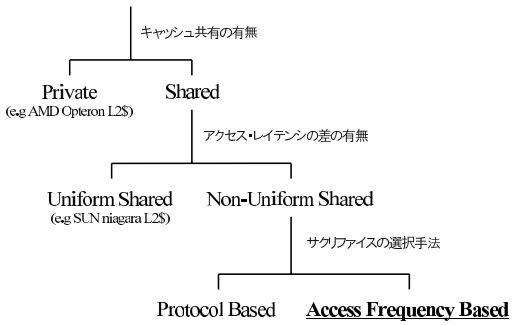


図 1 キャッシュ構成の分類
Fig. 1 Classification of cache.

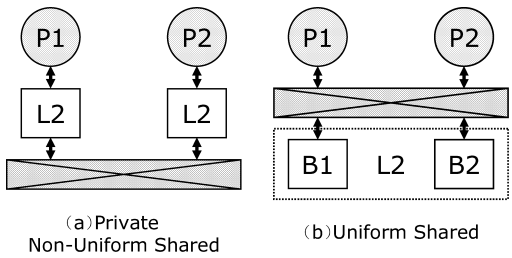


図 2 物理構成
Fig. 2 Physical structure.

側に複製して保持することが考えられる．しかし，低次キャッシュにおいてリプレースが行われた場合，ラインの情報をこの複製されたタグに毎回反映させる必要があり，構造が複雑化する．

このように，現在の一般的なプロセッサでは exclusive 構成をとることの利点はほとんどなく，欠点のみが存在するといえる．したがって，本稿では以降，キャッシュ階層間は inclusive 構成をとるものとして議論を行う．

2.1 private 構成と shared 構成

図 1 にマルチコア・プロセッサにおける L2 キャッシュ構成の分類を示す．マルチコア・プロセッサにおける L2 キャッシュの構成には，代表的なものとして **private** 構成と **uniform shared** 構成がある．**図 2** にこれらの物理構成を示す．

private 構成と **uniform shared** 構成は，アクセス・レイテンシと容量効率について，それぞれ相反する性質を持っている．この性質の違いは，両構成の物理構成における相互結合網の位置によって決まる．

private 構成では，それぞれのコアにキャッシュが直接接続されており，相互結合網はコアから見てキャッシュの向こう側にある．オンチップ・キャッシュ全体のうち，各コアから参照できるのは自身に接続されているキャッシュのみとなる．また，複数のコアで同じ領域を参照した場合，各コアのキャッシュにコピーをとる必要があるため，容量効率が低下する．

uniform shared 構成では，それぞれのコアは相互結合網を介して単一のキャッシュと接続されている．この単一のキャッシュは，アクセス・レイテンシを短くするため，複数のバンクに分けて実装されるのが普通である．このとき，各バンクは相互結合網を介してコアと接続されるため，アクセス・レイテンシは一様となる．

uniform shared 構成の場合，各コアはオンチップ・キャッシュ全体を共有することができる．しかし，キャッシュのどの部分も相互結合網を介してアクセスを行う必要があるため，private 構成と比較してアクセス・レイテンシが増加してしまうという問題がある．

2.2 uniform shared 構成と non-uniform shared 構成

private 構成と uniform shared 構成に対し，これらの中間の性質を持つのが **non-uniform shared** 構成である．non-uniform shared 構成は，図 2 (a) で示すように，private 構成と同様の物理構成をとる．

non-uniform shared 構成では，各バンクは物理的に近い位置にあるコアへ直接接続を行う．したがって，自身に接続されているバンクに対しては高速にアクセスを行うことができる．

以下に，non-uniform shared 構成におけるキャッシュ・アクセス時の動作を示す．

(1) 自身に接続されているバンクに対しアクセスを行う．ヒットした場合，従来のキャッシュと同様にラインをフェッチする．
(2) キャッシュ・ミスが発生した場合，次に他のコアのバンクに対してアクセスを行う．他のコアに目的のラインが存在した場合，そこからラインをフェッチする．なお，このとき，フェッチ元ラインの無効化は行わず，ラインはフェッチ先にコピーがとられる．
(3) 他のコアにおいても目的とするラインが存在しなかった場合，チップ外メモリからフェッチを行う．

これにより，自身に接続されているバンクに対しては private 構成と同等のアクセス・レイテンシを保ちながら，同時に各コアでバンクを共有することによるキャッシュ・ヒット率の向上を達成することが可能となる．

2.3 他のコアに対するアクセス・レイテンシ

non-uniform shared 構成において，他のコアが持つラインをフェッチする場合，そのアクセス・レイテンシは，uniform shared 構成におけるキャッシュのア

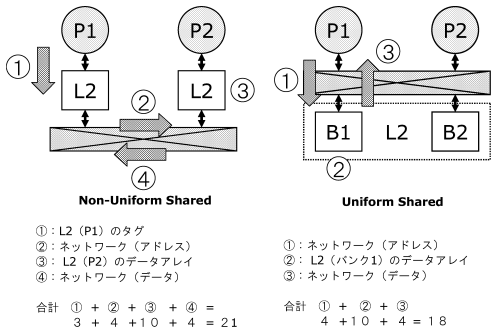


図 3 アクセス・レイテンシの比較
Fig. 3 Comparison of access latency.

クセス・レイテンシとほぼ同じである．それぞれの場合の動作を**図 3** に示す．

uniform shared 構成では，キャッシュからラインをフェッチする際，

(1) **アドレス要求**
(2) **データ・アレイへのアクセス**
(3) **データのフェッチ**

がクリティカル・パスとなる☆．このうち，アドレス要求とデータ・フェッチのアクセス・レイテンシは相互結合網を介する際の遅延によって決定される．

non-uniform shared 構成において，他のコアが持つラインをフェッチする場合，はじめに自身に直接接続されているキャッシュに対してヒット確認を行った後，uniform shared 構成と同様の手順を踏むことになる☆☆．ヒット確認はタグへのアクセスのみで行えるため，データ・アレイへのアクセスを待つ必要はない．したがって，両構成におけるアクセス・レイテンシの差は自身に接続されているキャッシュの，タグへのアクセス時間のみによって決定される．

一般に，L2 以上の高次キャッシュにおけるタグのアクセス・レイテンシはデータ・アレイのアクセス・レイテンシに比べて大幅に短い．これは，数 MB の大容量を持つキャッシュであっても，そのタグ・アレイは数十 KB と L1 キャッシュと同程度の大きさで構成できるためである．したがって，両構成におけるアクセス・レイテンシの差は小さいものとなる．

3. non-uniform shared cache の大域リプレースにおける既存の手法

non-uniform shared cache に対する既存の手法の多くでは，コヒーレンス・プロトコルにおけるラインの状態を基準として，**大域リプレース**と呼ぶ操作を行うことにより容量効率の向上を達成している．以下ではこの大域リプレースの動作について述べ，その後，ラインの状態を基準として大域リプレースを行うことの問題点を述べる．

☆ ここでは，キャッシュのタグとデータ・アレイは並列にアクセスするものと仮定している．
☆☆ 他のコアが持つラインをフェッチする場合なので，ヒット確認の結果は当然ミスヒットとなる．

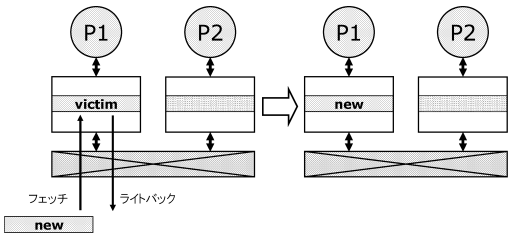


図 4 リプレース時の動作
Fig. 4 Replacing operation.

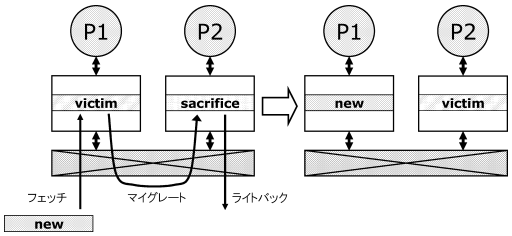


図 5 大域リプレース時の動作
Fig. 5 Global-replacing operation.

3.1 大域リプレース

通常のリプレースの動作を**図 4** に，大域リプレースの動作を**図 5** に示す．あるラインがリプレースの対象となった場合，他のコアに使用されていないライン・フレームがあれば，対象のラインをリプレースするのではなく，そのライン・フレームにマイグレートを行う．このとき，マイグレート対象のラインとマイグレート先のラインは，同じセットに属する必要がある．これにより，再度そのラインに対するアクセスがあったときには，チップ外のメモリからではなく，マイグレート先のキャッシュからラインをフェッチすることができるようになる．マイグレート先のキャッシュからのフェッチはオンチップで行われるため，そのアクセス・レイテンシはチップ外メモリへのアクセス・レイテンシに比べると非常に短く，全体としてアクセス・レイテンシを短縮することができる．以下では，マイグレート対象として選ばれたラインを**ヴィクティム（犠牲）**，マイグレート先として選ばれ，上書きされるラインを**サクリファイス**と呼ぶ．

3.2 プロトコル・ベースの手法による問題 サクリファイス選択の基準

大域リプレースを行う場合，マイグレート先となる

コアのキャッシュ・ヒット率に悪影響を及ぼさないように，サクリファイスを適切に選択しなければならない．

まず，invalid 状態のラインをサクリファイスとすることが考えられる．無効なラインは当然使用されていないので悪影響は生じない．先に述べた Terasawa らの提案する Z-Cache はこの考え方に基づき，無効なライン・フレームを有効活用することによって容量効率を向上させている．

Speight ら[2] は，invalid 状態のラインに加え，shared 状態のラインもサクリファイスとして選択する手法を提案している．shared 状態のラインはチップ上に複数存在する☆．したがって，shared 状態のサクリファイスが失われ，再びそのラインが必要となった場合でも，他のコアにあるラインを高速にフェッチすることができる．

上記の手法では，コヒーレンス・プロトコルにおけるライン状態に基づいて優先度を判断し，サクリファイスを決定している．以下，これらの手法を**プロトコル・ベースの手法**と呼ぶ．

問 題 点

上記のプロトコル・ベースの手法には，以下のような問題点が存在する．

(1) **選択できないことがある**
一般に，exclusive 状態のラインが占める割合は比較的高い．特に，各コア上で独立したプロセスが動作している場合等は，ほとんどのラインは exclusive 状態である．このような場合，invalid や shared 状態のラインのみをサクリファイスとして選択する手法では，そもそも選択対象が存在しない．

(2) **不適切なラインを選択することがある**
まったくアクセスされない exclusive なラインと頻繁にアクセスされる shared なラインがあった場合，当然前者をサクリファイスとすべきである．しかし，特定状態のラインのみをサクリファイスとする手法では，後者を選択してしまう．

4. LRU を用いた大域リプレース

前述したように，プロトコル・ベースの手法ではラインの状態を基準としているため，サクリファイスの選択対象がない場合や，不適切なサクリファイスを選択してしまうという問題がある．

そこで，我々は **LRU**（least-recently used）を用いたキャッシュ・ラインの大域置き換えを提案する．チップ全体の同一セットに属するラインに対し，単一の**大域 LRU 情報**を管理し，参照して大域リプレースを行う．これにより，ラインの状態ではなく，ラインのアクセス頻度に基づいてサクリファイスの選択を行う．

大域 LRU 情報を参照して大域リプレースを行う場合の動作を以下に示す．

(1) 通常のリプレース時と同様にしてヴィクティムを決定する．すなわち，リプレースが発生したコアのキャッシュにおいて LRU となるラインを選択し，ヴィクティムとする．このヴィクティムの決定には，各コアがそれぞれ独立に持つ LRU 情報が使用される．

(2) 大域 LRU 情報を参照し，同一セットに属するチップ全体のラインの中で LRU となるラインを選択してサクリファイスとする．

(3) サクリファイスが存在したライン・フレームに対し，ヴィクティムのマイグレートを行う（大域リプレース）．なお，リプレースが発生したコアで LRU となるラインと大域 LRU 情報によって選択されたサクリファイスが一致してしまった場合，そのラインは従来と同様にリプレースされる．

4.1 手法の利点

大域 LRU 情報を参照して大域リプレースを行うことにより，アクセス頻度の低いライン・フレームをサクリファイスとして選択することができる．アクセス頻度に従ってサクリファイスを選択することにより，特に以下のような性能向上が得られる．

(1) **ワーキングセットの大きさに差がある場合**
ワーキング・セットの小さいスレッドが動作しているコアでは，使用されていない余剰なキャッシュが存在している．この余剰なキャッシュはアクセス頻度の低いラインとなり，優先的にサクリファイスとして選択される可能性が高い．したがって，そのようなコアの余剰なキャッシュが，あたかも他のコアの L3 であるかのように機能する．

(2) **コア数より稼動スレッドが少ない場合**
休止状態のコアが存在した場合，そのコアが持つキャッシュはアクセスが行われない．したがって，そのようなコアのラインは優先的にサクリファイスとして選択される．これにより，休止状態のコアが持つキャッシュ全体が，他のコアに対して L3 キャッシュであるかのように機能する．スレッド間同期や IO の同期待ちのために，局所的に休止状態となるスレッドが存在した場合にも，本手法は有効である．

☆ 一般に，shared 状態のラインは必ずしも複数存在しないが，複数存在する可能性は高い．

4.2 擬似大域 LRU

本提案手法をそのまま実装した場合，各コアが持つすべてのラインに対して LRU アルゴリズムを構成することとなる．LRU アルゴリズムを構成するためのコストは，要素数に応じて急激に大きくなるため，各コアが持つすべてのラインに対して LRU アルゴリズムを適用することは現実的ではない．したがって，以下に示す擬似大域 LRU アルゴリズムを使用する．

一般に，LRU アルゴリズムを実現する機構は，キャッシュにアクセスがあるごとに，どのラインに対してアクセスが行われたかを通知され，内部状態を更新する．この動作は擬似大域 LRU 機構においても，基本的には同じである．

ここで，大域 LRU 情報を提供する機構に対して行われるアクセスの通知を，ライン単位ではなく，各コア単位で行うように変更する．すなわち，セットごとに大域 LRU 情報を保持するものの，各コアにおいて同一セット内のどのラインにアクセスが行われたかということに関しては区別を行わないようにする．したがって，あるラインに対してアクセスが行われた場合，大域 LRU 機構には，そのコアのインデックスを通知する．

このコア単位の LRU 情報と，各コアが元々持っているキャッシュの LRU 情報を組み合わせることで，擬似的な大域 LRU 情報を構成する．サクリファイス決定時は，まずこのコア単位の LRU 情報を参照し，どのコアに属するキャッシュが LRU となるかを特定する．次に，選択されたキャッシュが通常のリプレース時に使用する LRU 情報を利用し，その中で LRU となるラインをサクリファイスとする．

これにより，擬似大域 LRU 情報を構成するために必要な要素数は，各セットにおけるチップ全体のライン数から，チップ内におけるコアの数に減少する．したがって，現実的なコストで大域 LRU 情報を構成することが可能になる．

4.3 参照履歴によるスワップ

大域置き換えによる容量効率を向上させるため，以下ではキャッシュの各ラインに対し，状況に応じてラインのスワップを行う手法について述べる．

無駄なコピーの発生

ヴィクティムが大域リプレースにより，他のコアのキャッシュに退避された後，再び元のコアにフェッチされる場合を考える．このとき，退避先のキャッシュと自身のキャッシュには同一ラインのコピーがとられる．

各コアで独立したプロセスが稼動している場合，特定のコアで使用するラインは，そのコアでのみ使用される場合が多い．したがって，このような場合，退避先のキャッシュには使用されることのない無駄なコピーが残されることになり，キャッシュの容量効率を下げる一因となる．

表 1 大域 LRU 情報を検討する際の L2 キャッシュ・パラメータ
Table 1 L2 cache parameter for discussion of global LRU information.

サイズ	2 MB
ライン・サイズ	64 B
連想度	8

ラインのスワップ

無駄なコピーが発生することを避けるため，状況に応じてラインのスワップ動作を行う．すなわち，他のコアからラインの再フェッチを行った際，そのラインがそのコアでしか使われていなかった場合は，大域 LRU 情報を参照して大域リプレースを行うのではなく，そのフェッチ先のラインが存在したフレームにヴィクティムをマイグレートする．

スワップを行うかどうかの判定は，それぞれのキャッシュのラインに，自身が接続されているコアからのアクセスが行われたことを示すビットを追加して行う．このビットは自身が接続されているコアからアクセスを受けた場合に有効になり，大域リプレースによって他のコアからラインがマイグレートされた際には無効化される．他のコアからフェッチを行う際は，このビットを同時にフェッチし，ビットがクリアされていた場合はフェッチ先のコアにラインを残す必要はないとして，ラインのスワップを行う．

ハードウェア・コスト

ラインのスワップを行うために必要となるハードウェアのうち，主なものはアクセスが行われたことを示す情報を保持するメモリである．先にも述べたように，このメモリはキャッシュのラインごとに 1 ビット必要となる．**表 1** に示すキャッシュ構成をとった場合，ラインの数は 32768 となる．したがって，この場合，32768 ビット（4 KB）のメモリが新たに必要となる．2 MB のデータ・アレイに対して 4 KB のメモリは十分に少さく，したがって，ラインのスワップを行うために必要となるハードウェアは比較的小さいものであるといえる．

4.4 大域 LRU 機構の実装

大域 LRU 機構は，各コアで共有される単一の機構である．したがって，大域 LRU 機構へのアクセスにはある種の調停機構を通す必要があり，そのアクセス・レイテンシとバンド幅が問題となりうる．以下では，このアクセス・レイテンシとスループットについて検

討を行う．

アクセス・レイテンシ

大域 LRU 機構に対するアクセスは，ラインのフェッチ時における LRU 情報の更新と，リプレース時における LRU 情報の取得である．この 2 つは，ラインをフェッチする際のクリティカル・パス上には存在しない．また，この LRU 情報はあくまでリプレース時のヒントであり，更新の遅れを許したとしても，プログラムの実行の正しさには影響しない．したがって，LRU 情報の更新と参照にある程度のレイテンシが存在したとしても問題はないと考えられる．

スループット

RAM で状態を保持することによって LRU アルゴリズムを実装することを考える．先に述べた擬似大域 LRU アルゴリズムを用いた場合，共有される情報はコアの数分だけでよい．したがって，たとえば 4 コアのマルチコア・プロセッサの場合，24 通り（5 ビット）の状態を保持する必要がある．表 1 に示すキャッシュの構成をとった場合，セットの数は 4096 となる．したがって，LRU 情報を構成するのに必要な RAM の容量は合計で 20480 ビット（2.5 KB）となる．これは分岐予測に用いられる PHT（pattern history table）等と同程度の大きさであり，1 サイクルでアクセス可能であると考えられる．

LRU 情報の更新は，この RAM に対して read-modify-write のシーケンスを経て行われる．それぞれのステージは 1 サイクルで処理可能なものとすると，パイプライン動作により，1 サイクルあたり 1 つのアクセス情報を更新可能となる．

LRU 情報へのアクセスは複数のコアから行われるため，共有される情報へのアクセスには調停機構を通す必要がある．複数のコアからのアクセスに競合が発生した場合，競合したアクセス要求は待ち行列に入れられる．大域 LRU 機構へのアクセスは，各コアのキャッシュ・アクセスの際に発生し，コアの数だけ発生する．L2 以降の高次キャッシュの場合，各コアのキャッシュへのアクセス頻度は低く，上記のモデルに従って 1 サイクルあたり 1 つのアクセスが処理できるのであれば，アクセス競合の影響も比較的少ないものと予想される．このアクセス競合については，後述する評価の章において，実際にどの程度発生するかをシミュレーションにより確かめた．

5. 評　　価

5.1 評価方法

システム・シミュレータである Simics [3] に対して提案手法を実装し，評価を行った．Simics はプロセッサだけでなく，各種デバイスのシミュレーションも行うため，OS を含めたアプリケーションの動作をシミュレーションすることが可能である．提案手法の実装については，先に述べた擬似大域 LRU を用いたものと，あわせてスワップを行うものについて実装を行った．また，性能向上の理論値を測定するため，完全な大域 LRU を用いた実装も行い，比較を行った．

表 2 各構成における記憶階層ごとのレイテンシ
Table 2 Access latency of each memory hierarchy.

モジュール	アクセス・レイテンシ
Private 2 MB 8 Way	
L2	10 cycle
uniform shared 8 MB 32 Way	
L2	18 cycle
non-uniform shared 2 MB 8 Way	
L2（ローカル）	10 cycle
L2（他のコア）	21 cycle
共通	
L1（2-way，64 B-line）	1 cycle
チップ外メモリ	150 cycle

他の比較対象としては，private 構成，uniform shared 構成，プロトコル・ベースの non-uniform shared 構成についても実装を行い，評価した．プロトコル・ベースの non-uniform shared 構成については，先に述べた Terasawa らの手法に相当する invalid 状態のラインを利用するものと，Speight らの手法に相当する shared 状態のラインも加えて利用するものを実装した．なお，この shared 状態のラインを利用する実装では，サクリファイスの候補として invalid 状態のラインと shared 状態のラインが同時に存在した場合，Speight らが行ったのと同様に invalid 状態のラインが優先して使用されるように実装した．

構　　成

IA32 命令セットを持つマルチコア・プロセッサ（4-Core）のシミュレーションを行った．各コアは out-of-order 実行可能なプロセッサ（最大 3 命令同時発行）であり，L1 キャッシュとして各 16 KB の命令キャッシュとデータ・キャッシュを搭載している．また，**表 2** に，各記憶階層における構成とアクセス・レイテンシを示す．L2 キャッシュのアクセス・レイテンシは，2.3 節での議論をもとに，**表 3** に示すパラメータを使用して決定した．

ベンチマーク

以下に示す 4 種類のベンチマークを用いて評価を行った．全コア合計で 16×10^6 命令のウォームアップ実行を行った後，全コア合計で 1.2×10^9 命令の実行を行い，評価を行った．コンパイルが必要なものに関

表 3 L2 キャッシュにおけるパラメータ
Table 3 L2 cache parameter.

モジュール	アクセス・レイテンシ
タグ	3 cycle
データ	10 cycle
ネットワーク	4 cycle

表 4 SPLASH-2 ベンチマークのパラメータ
Table 4 SPLASH-2 parameter.

アプリケーション	パラメータ
LU	-p4 -n1024
OCEAN	-p4 -n514

表 5 SPEC2000 アプリケーションの組合せ
Table 5 SPEC2000 Mix.

組合せ	アプリケーション
Mix-A	aspi,art,equake,mesa
Mix-B	ammp,mesa,swim,vortex
Mix-C	apsi,gzip,mcf,mesa
Mix-D	ammp,gzip,vortex,wupwise

しては gcc 3.4.2 を用いてコンパイルを行い，ターゲット上のオペレーティング・システムとしては Fedora Core3（Linux Kernel 2.6.9-1.667 smp）を使用した．

(1) **SPECjbb2005** SPECjbb2005[4] は，Java により実装された，典型的なビジネス・アプリケーションの動作をシミュレートするベンチマークである．4 warehouse 分のシミュレーションを行い，評価を行った．Java の実行環境としては Sun Microsystems JRE 1.5.0_06 を用いた．

(2) **OSDL-DBT2** Open Source Development Labs Database Test 2 ver 0.37[5] はオンライン・トランザクションの性能を評価するベンチマークであり，複数の作業者が 1 つのデータベースにアクセスする卸売業者のシステムをシミュレートする．作業者の思考時間とキー押下時間を 0 とし，3 warehouse 分のデータを使用して評価を行った．データベース・エンジンには MySQL 4.1 を用いた．

(3) **SPLASH-2** SPLASH-2[6] より，LU と OCEAN を用いて評価を行った．各アプリケーションにおける実行時パラメータを**表 4** に示す．

(4) **SPEC2000** SPEC2000[7] より複数のアプリケーションを組み合わせて実行した．**表 5** にその組合せを示す．なお，この組合せは 6 章で述べる Chishti ら[8] の研究で評価に用いられたものと同一である．

5.2 評価結果

すべてのベンチマークにおける uniform shared 構成からの IPC 向上率の平均値を**図 6** に示す．また，各ベンチマークにおける IPC 向上率の内訳を**図 7** に示す．各構成とグラフ中のラベルの対応は**表 6** のとおりである．

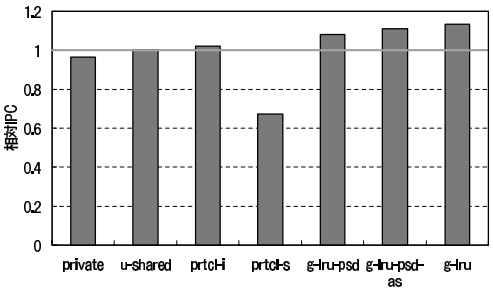


図 6 平均 IPC 向上率
Fig. 6 Average IPC improvement ratio.

表 6 ラベルとキャッシュ構成の対応
Table 6 Label and each method.

ラベル	キャッシュ構成
private	private 構成
u-shared	uniform shared 構成
prtcl-i	invalid なラインを用いる手法
prtcl-s	invalid に加え，shared なラインも用いる手法
g-lru-psd	擬似大域 LRU を用いた提案手法
g-lru-psd-as	擬似大域 LRU とスワップを用いる提案手法
g-lru	完全な大域 LRU とスワップを用いる提案手法

グラフからは，以下のことが読み取れる：

- g-lru-psd-as は u-shared に対し，平均で 11%の性能向上を達成している．また，prtcl-i に対しては 8.6%の性能向上を達成している．
- 既存手法はベンチマークによって得手不得手が見られるのに対し，提案手法は各既存手法の最良と同等かそれ以上の性能を達成している．
- prtcl-s は全体にわたり，大きく性能を下げている．
- g-lru-psd-as と g-lru-psd を比較した場合，2.6%程度の性能向上が見られる．特に，SPLASH2-LU では 6.6%，SPEC2000 Mix-C では 10%と大きく性能が向上している．
- g-lru-psd-as と g-lru を比較した場合，平均して 1.2%程度の性能低下が起きている．しかし，SPEC2000 においてはあまり性能は変化せず，Mix-A においては若干逆転が見られる．

L2 キャッシュ・ヒット率

図 8 と**図 9** に各ベンチマーク実行時の L2 キャッシュ・ヒット率を示す．ここで，L2 キャッシュ・ヒット率とは，全コアにおける L1 キャッシュ・ミス時に生じた L2 キャッシュへのアクセスの総計を分母として，L2 キャッシュにヒットしたアクセス数の割合を表

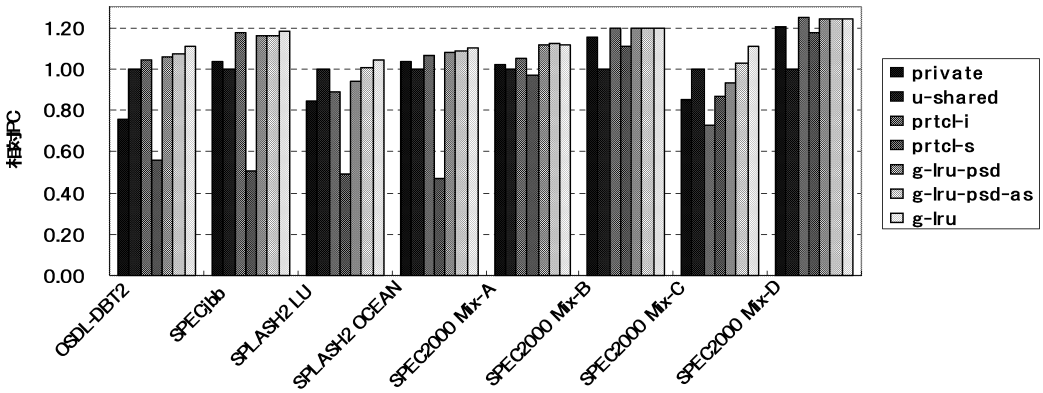


図 7 IPC 向上率
Fig. 7 IPC improvement ratio.

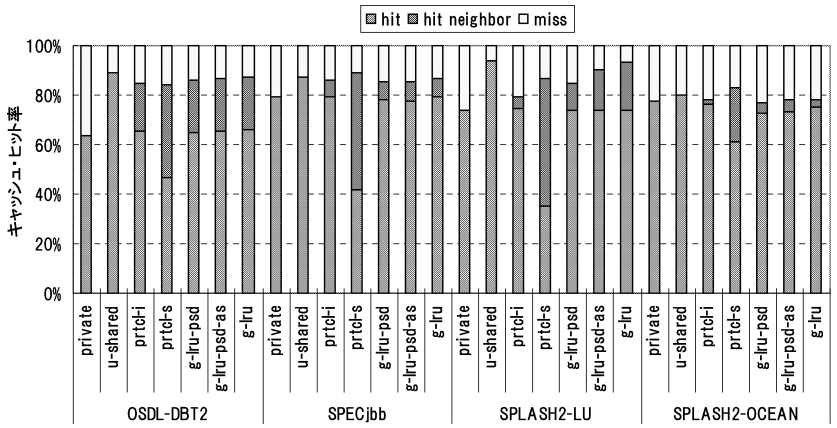


図 8 L2 キャッシュ・ヒット率（group-mt）
Fig. 8 L2 cache hit ratio (group-mt).

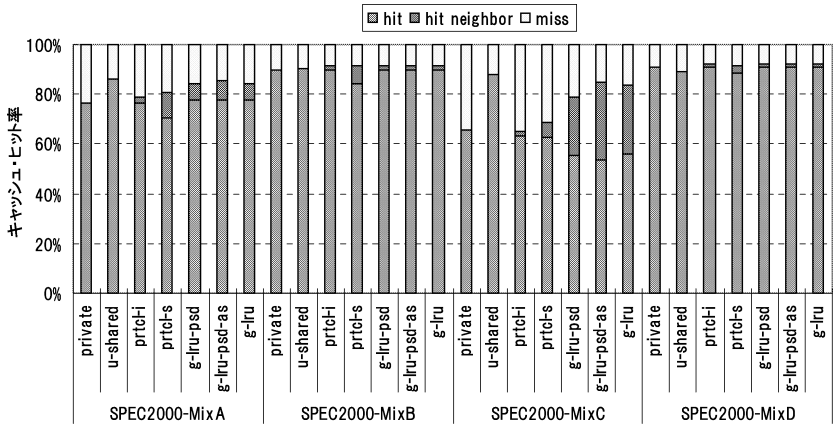


図 9 L2 キャッシュ・ヒット率（group-mp）
Fig. 9 L2 cache hit ratio (group-mp).

す．各図において，hit は private なキャッシュに対するヒット，hit-neighbor は他のコアが持つキャッシュに対するヒット，そして miss はチップ外メモリへのアクセスが発生した場合の割合を表す．

グラフからは，以下のことが読み取れる：

- OSDL-DBT2，SPECjbb，SPLASH2 のベンチマーク集合（以下，これらをまとめて **group-mt** とする）では，prtcl-s における hit-neighbor の割

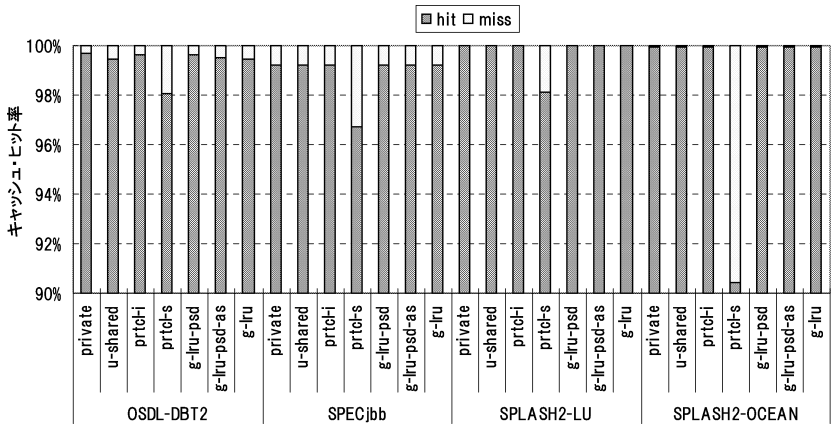


図 10 L1 命令キャッシュ・ヒット率（group-mt）
Fig. 10 L1 instruction cache hit ratio (group-mt).

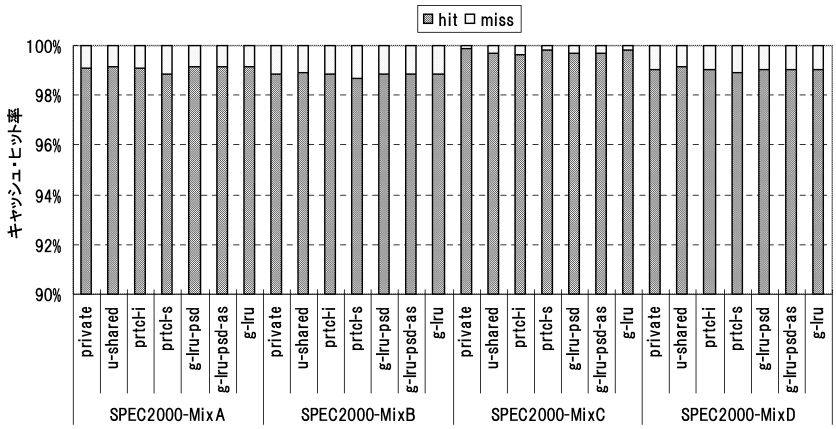


図 11 L1 命令キャッシュ・ヒット率（group-mp）
Fig. 11 L1 instruction cache hit ratio (group-mp).

合が非常に大きくなっている．一方，SPEC2000 を組み合わせて実行したベンチマーク集合（以下，これらをまとめて **group-mp** とする）では，増加は比較的小さい．この hit-neighbor の増加は，平均的なアクセス・レイテンシを増加させるため，性能を下げる一因となる．

- SPEC2000-MixA と SPEC2000-MixC では，prtcl-i と prtcl-s が private と同程度のヒット率であるのに対し，提案手法では u-shared と同等のヒット率を達成している．
- OSDL-DBT2 や SPECjbb では，u-shared に対して g-lru-psd-as のミス率はそれぞれ 3%と 1.5%大きくなっている．しかし，g-lru-psd-as の L2 キャッシュ・アクセスの大部分は private なキャッシュであるため，u-shared と比較してヒット時の平均的なアクセス・レイテンシは大幅に短い．この結果，g-lru-psd-as は u-shared より高い性能を示しているのであると考えられる．

group-mt では各プロセスにおいて複数のスレッドが実行されており，invalid や shared 状態のラインが一定量存在する．これに対し，group-mp では複数の独立したプロセスが実行されており，ラインの大部分は exclusive 状態である☆．このため，group-mp では prtcl-i や prtcl-s において大域リプレースがあまり行われず，結果として，SPEC2000-MixA と SPEC2000-MixC で提案手法のみが大きなヒット率の改善を見せたのであると考えられる．

低次キャッシュに対する影響

図 10 と**図 11** に各ベンチマーク実行時の L1 命令キャッシュ・ヒット率を示す．また，**図 12** と**図 13** に各ベンチマーク実行時の L1 データ・キャッシュ・ヒッ

☆ OS や共有ライブラリが存在するため，exclusive 状態以外のラインも少量存在する．

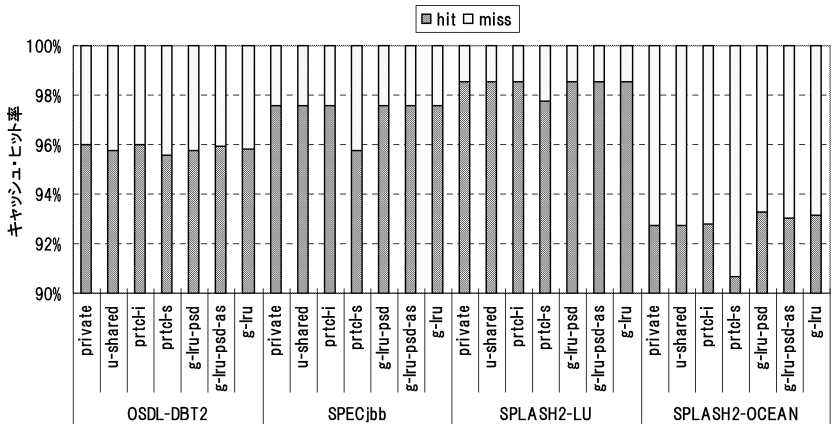


図 12 L1 データ・キャッシュ・ヒット率（group-mt）
Fig. 12 L1 data cache hit ratio (group-mt).

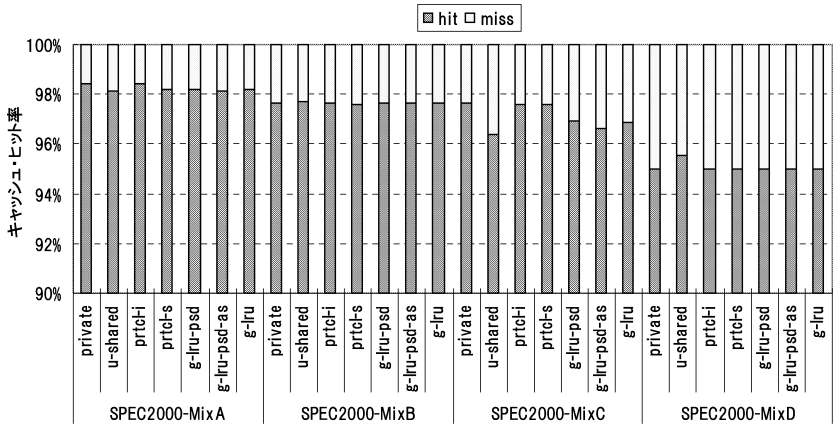


図 13 L1 データ・キャッシュ・ヒット率（group-mp）
Fig. 13 L1 data cache hit ratio (group-mp).

ト率を示す．ここで，L1 キャッシュ・ヒット率とは，全コアにおける L1 キャッシュへの総アクセス数を分母として，各コアで L1 キャッシュにヒットしたアクセス数の割合を意味する．

グラフより，group-mt において prtcl-s のみ，ミス率が大幅に増加しているのが分かる．特に命令キャッシュにおけるミス率の増加が著しく，先に述べた prtcl-s における性能低下の主な原因となっている．これに対し，group-mp では特にミス率の増大は見られない．

このように prtcl-s においてヒット率に差が生じるのは，先に述べたのと同じく，group-mt と group-mp の間でアクセス・パターンが異なるためである．group-mp では shared 状態のラインが少ないため，サクリファイス対象があまり存在せず，結果として低次キャッシュの無効化が生じにくくなっている．

提案手法では prtcl-s と同様に，shared 状態のラインもサクリファイスの対象とするため，低次キャッシュのラインを無効化してしまう可能性がある．しかし，g-lru-psd-as では prtcl-s で生じたようなミス率の増加は見られない．これは，高次キャッシュにおいてアクセス頻度の高いラインは低次キャッシュ上にも存在する確率が高いため，高次キャッシュのアクセス頻度に基づきサクリファイスを選択することで，低次キャッシュに存在するラインの無効化を回避しているのであると考えられる．

ネットワーク・レイテンシと性能

L2 キャッシュのアクセス・レイテンシを決定する際に用いたパラメータ（表 3）のうち，ネットワーク・レイテンシについてパラメータを変えて評価を行った．SPECjbb と SPEC2000-MixA についての結果を**図 14** と**図 15** に示す．u-shared と prtcl-s がネットワーク・レイテンシの増加とともに性能を落としてい

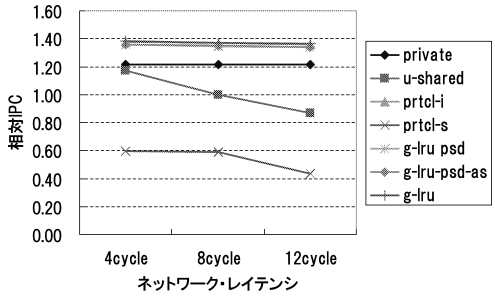


図 14 ネットワーク・レイテンシと相対 IPC（SPECJBB）
Fig. 14 Network latency and relative IPC (SPECJBB).

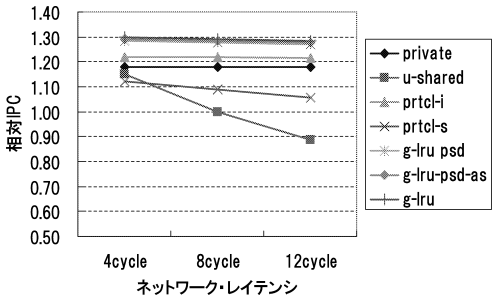


図 15 ネットワーク・レイテンシと相対 IPC（SPEC2000-MixA）
Fig. 15 Network latency and relative IPC (SPEC2000-MixA).

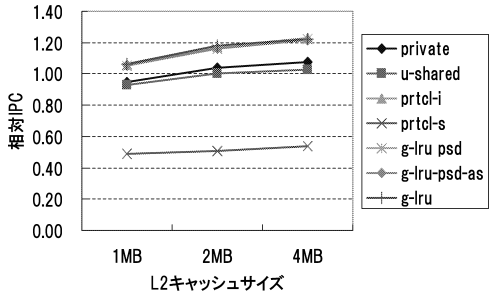


図 16 キャッシュ・サイズと相対 IPC（SPECJBB）
Fig. 16 Cache size and relative IPC (SPECJBB).

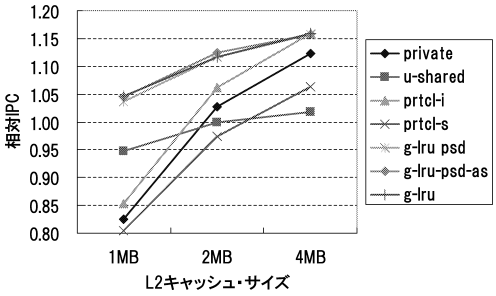


図 17 キャッシュ・サイズと相対 IPC（SPEC2000-MixA）
Fig. 17 Cache size and relative IPC (SPEC2000-MixA).

るのに対し，g-lru-psd-as ではネットワーク・レイテンシの影響をあまり受けていないことが分かる．これは，g-lru-psd-as ではネットワーク・レイテンシの影響を受ける，他のコアの持つキャッシュへのアクセスの割合が小さいためである．図 8 と図 9 より，g-lru-psd-as における他のコアの持つキャッシュへのアクセスの割合は，すべてのベンチマークにおいて同様に小さいため，他のベンチマークについても同様の傾向をとる．

キャッシュ・サイズと性能

L2 キャッシュのサイズについて，パラメータを変えて評価を行ったものを**図 16** と**図 17** に示す．

図 8 と図 9 より，各ベンチマークは u-shared と提案手法のみがヒット率が高いグループ（SPLASH2-LU，SPEC2000-MixA，SPEC2000-MixC）と，各方式で大きなヒット率の差が存在しないグループが存在するため，それらの中から SPEC2000 Mix-A と SPECjbb について評価を行った．

SPECjbb（図 16）では，容量の変化に対して各構成とも同程度の性能の変化を見せている．SPEC2000 Mix-A（図 17）では，u-shared と g-lru-psd-as がキャッシュ容量の減少にともない，比較的ゆるやかに性能が低下しているのに対し，その他の既存の構成では大きく性能を下げている．これは，SPEC2000 Mix-A では既存の各構成と比較して提案手法のみが u-shared に近い L2 キャッシュ・ヒット率を達成しているためである．

スワップを行った場合の性能向上

g-lru-psd と g-lru-psd-as を比較した場合，平均で 2.6%の性能向上が得られており，SPLASH2-LU と SPEC2000 Mix-C において，特に大きな性能向上が見られた．図 8 と図 9 より，g-lru-psd と比較して g-lru-psd-as では，SPLASH2-LU において 5.8%，SPEC2000 Mix-C において 5.5%ヒット率が改善されている．両ベンチマークにおける IPC の向上は，この L2 キャッシュ・ヒット率の改善によるものであると考えられる．

他のベンチマークについては，特に大きなヒット率の改善は見られないものの，すべてのベンチマークにおいて g-lru-psd-as は g-lru-psd より良いヒット率を示している．4.3 節で述べたように，スワップを行うために必要となるハードウェア・コストは十分に小さく，この性能向上は有意であるといえる．

擬似 LRU による性能低下

g-lru-psd-as と g-lru を比較した場合，平均して 1.2%程度の性能低下が起きており，特に SPLASH2-LU と SPEC2000-MixC では比較的大きな性能低下

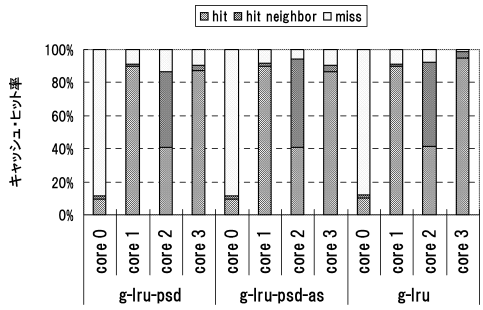


図 18 コアごとの L2 キャッシュ・ヒット率（SPEC2000-MixC）
Fig. 18 L2 cache hit ratio on each core (SPEC2000-MixC).

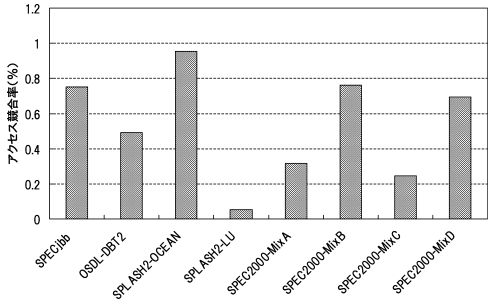


図 19 大域 LRU 機構へのアクセス競合率
Fig. 19 Access contention ratio in global LRU unit.

が起きている．

図 8 より，SPLASH2-LU では g-lru のミス率が 6.5%であるのに対し，g-lru-psd-as ではミス率が 9.7%とミス率が約 1.5 倍に増加しているため，このことが性能差につながっているものと考えられる．

図 18 に，SPEC2000-MixC における，コアごとの L2 キャッシュ・ヒット率を示す．g-lru-psd-as と g-lru を比較すると，g-lru では core0 から core2 において，それぞれヒット率が約 2%低下しているのに対し，core3 ではヒット率が改善され，ミスが 1.4%とほとんどなくなっている．これにより，チップ全体としてのヒット率はあまり変わらないものの，core3 における性能が大きく向上している．

g-lru が理想的な LRU を用いて実装されているのに対し，g-lru-psd-as は 4.2 節で述べた擬似 LRU を用いて実装されているため，真に LRU であるラインをサクリファイスとして選択しない場合がある．このため，上記のようなヒット率の違いが生じているのであると考えられる．

大域 LRU 機構へのアクセス競合

4.4 節のモデルにおいて，調停機構に対して仮想的に長さ無限の待ち行列を配置し評価を行った．その待ち行列の中に要素が存在した時間（サイクル数）の割合を**図 19** に示す．

グラフからは，アクセスの競合が発生している時間は最大でも 1%以内であることが分かる．したがって，アクセス競合が発生した際に，チップ全体をストールさせるような単純な実装を行ったとしても，性能の低下はたかだか 1%以内である．

6. 関 連 研 究

COMA

non-uniform shared cache と同様の考え方は，かつての **COMA** [9] においても見ることができる．COMA（Cache Only Memory Architecture）は，共有メモリ型並列計算機のためのメモリ・アーキテクチャである．COMA は，それ以前の ccNUMA（Cache-Coherent Non-Uniform Memory Architecture）に対して，主記憶の容量効率を向上させたものと考えることができる．ccNUMA には，通常 DRAM で構成される各ノードの主記憶の一部を，リモート・ノードの主記憶に対するキャッシュとして利用できるものがある．その場合，同じ主記憶上にあるコピーであっても，もとからそのノードにあったメモリ・コピーであるか，リモートからキャッシュしてきたキャッシュ・コピーであるかの区別が存在する．そして，その要/不要にかかわらず，メモリ・コピーをノードから削除することはできない．それに対して COMA では，メモリ・コピー/キャッシュ・コピーの区別をなくし，主記憶の全域をキャッシュとして利用できるようにしたものである．

COMA におけるノード間のコピーのやりとりの様子は，これまで述べてきた non-uniform shared cache のそれと基本的に同じと考えてよい（ただし COMA では，ヴィクティムがシステム内で唯一のコピーであるときには，それを単にリプレースするということはできない．non-uniform shared cache では，別にメモリ・コピーがあるので，clean であればリプレースできる）．共有の度合い（degree of share）を最適化することによって，容量効率とアクセス・レイテンシをともに向上させるという点で，non-uniform shared cache と COMA の目的は一致している．同じ考え方を，マルチプロセッサの主記憶に適応したものが COMA，マルチコア・プロセッサのオンチップ・キャッシュに適応したものが non-uniform shared cache であるといえる．しかし，以下に述べるように，この適用する対象の違いは重要である．

まず第 1 に，対象とする資源の価値が大きく異なる．マルチプロセッサの主記憶は，COMA 化による制御

の複雑化に見合うほど高価な資源とはいえない．また，（COMA ではない）ccNUMA であっても，アフィニティ制御によって不要なメモリ・コピーの発生をソフトウェア的に抑制することができる．このことは，現在の共有メモリ型マルチプロセッサにおいて COMA が主流となっていない理由と考えられる．一方，マルチプロセッサの主記憶に比べて，マルチコア・プロセッサのオンチップ・キャッシュは非常に高価な資源である．すなわち，共有の度合いを最適化することによって容量効率とアクセス・レイテンシをともに向上させるという考えは，COMA ではなく，non-uniform shared cache にこそ適しているといえる．

第 2 に，ノード間の通信コストが大きく異なる．COMA における各主記憶は別チップ（別モジュール）とならざるをえず，ノード間の通信コストは非常に大きい．そのため，共有の度合いを最適化するには大きなコストがかかる．これに対して non-uniform shared cache の場合，オンチップゆえ，通信コストは非常に小さい．各ノード間で LRU 情報を共有するという本稿の提案は，COMA ではほとんど実現不可能であり，non-uniform shared cache ならではのものであるということができる．

NUCA をベースとした研究

マルチコア・プロセッサの non-uniform shared cache の研究としては，シングルコア・プロセッサの **NUCA**（Non-uniform cache architecture）[10] をベースとした研究が多く行われてきた．NUCA では，コアからの物理的距離に応じて，近い位置にあるバンクほど短いレイテンシでアクセス可能な構造となっている．アクセス頻度の高いラインをコアの近くにあるバンクにマイグレートしていくことで，キャッシュ全体のアクセス・レイテンシを短縮している．

Beckmann ら[11] は，この NUCA をマルチコア・プロセッサに適用した手法を提案している．Bradford らの手法では，あるラインがアクセスされた場合，アクセスを行ったコアの近くにラインがマイグレートされる．Beckmann らの手法ではラインのコピーが許されないため，複数のコアから参照されるデータが存在すると，各コアでラインの奪い合いが起きてしまう問題が存在する．ラインの奪い合いが生じると，そのラインはアクセスを行っている複数のコアの中間位置に配置されてしまい，アクセス・レイテンシが増大してしまう．

Chishti ら[8] はこれらの考え方をさらに発展させた CMP-NuRAPID を提案している．CMP-NuRAPID ではタグとラインを双方向ポインタで結び付けて管理することにより，ラインのマイグレートや複製を行っている．CMP-NuRAPID ではラインの複製を行うことが可能なため，上記のようなラインを奪い合う問題はない．しかし，タグとデータ・アレイの双方にポインタを格納するためのコストは大きく，また，これらを使用・管理するリプレース・アルゴリズムは非常に複雑である．このため，この機構を実際のハードウェアに実装するのは非常に困難であると考えられる．

7. おわりに

本稿ではマルチコア・プロセッサの non-uniform shared cache において，ラインの大域 LRU に基づいてサクリファイスを選択し，大域リプレースを行う手法について述べた．シミュレーションによる評価の結果，既存のキャッシュ構成と比較して平均で 8.6%の性能向上を達成した．

本稿で述べた手法は，通信コストに対する制約が緩和されたマルチコア・プロセッサだからこそ成り立つものである．しかし，コア数が大きく増大した場合，マルチコア・プロセッサでもプロセッサ内部の通信コストが問題となることが予想される．たとえば，大域 LRU 機構へのアクセスのバンド幅が問題となりうる．本稿で評価を行った 4 コアのマルチコア・プロセッサの場合，大域 LRU 機構へのアクセス競合は，シミュレーションによって十分に小さいことが確認された．しかし，数十のコアを持つマルチコア・プロセッサに対して提案手法を適応した場合，そのバンド幅が不足することは確実である．したがって，本手法はある程度少数のコアによって構成されるマルチコア・プロセッサに適応するか，あるいは少数のコアどうしをクラスタリングして適応するべきであるといえる．今後はこのようにコア数が大きく増加した場合にもスケーラビリティを持つ手法について検討してみたい．

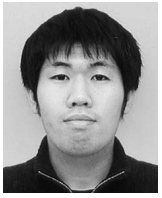
謝辞 本稿の研究は，一部 21 世紀 COE「情報技術戦略コア」，および科学技術振興機構 CREST「ディペンダブル情報処理基盤」による．

参考文献

1) Terasawa, T.: Z Cache: A Cache Mechanism for On-chip Multiprocessors, *Trans. IPSJ*, Vol.37, No.4, pp.666–669 (1996).
2) Speight, E., Shafi, H., Zhang, L. and Rajamony, R.: Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors, *Proc. 32nd International Symposium on Computer Architecture* (*ISCA*), pp.346–356 (2005).

3) Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. and Werner, B.: Simics: A full system simulation platform, *IEEE Computer*, Vol.35, No.2, pp.50–58 (2002).
4) The Standard Performance Evaluation Corporation: SPECjbb2005 (Java Server Benchmark). http://www.spec.org/jbb2005/
5) Open Source Development Labs: Open source development labs database test 2. http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/
6) Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. 22nd International Symposium on Computer Architecture* (*ISCA*), pp.24–36 (1995).
7) The Standard Performance Evaluation Corporation: Spec CPU2000 suite. http://www.spec.org/cpu2000/
8) Chishti, Z., Powell, M.D. and Vijaykumar, T.N.: Optimizing Replication, Communication, and Capacity Allocation in CMPs, *Proc. 32nd International Symposium on Computer Architecture* (*ISCA*), pp.357–368 (2005).
9) Hagersten, E., Landin, A. and Haridi, S.: DDM — A Cache-Only Memory Architecture, *IEEE Computer*, Vol.25, No.9, pp.44–54 (1992).
10) Kim, C., Burger, D. and Keckler, S.W.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, *Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), pp.211–222 (2002).
11) Beckmann, B.M. and Wood, D.A.: Managing Wire Delay in Large Chip-Multiprocessor Caches, *Proc. 37th annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO*), pp.319–330 (2004).

(平成 18 年 7 月 21 日受付)
(平成 18 年 12 月 10 日採録)



塩谷 亮太

1981 年生．2006 年東京大学工学部電子工学科卒業．東京大学大学院情報理工学系研究科修士課程に在学中．コンピュータアーキテクチャの研究に従事．

ルォン ディン フォン（学生会員）

1977 年生．2004 年東京大学大学院情報理工学系研究科修士課程修了．同研究科博士課程に在学中．コンピュータアーキテクチャの研究に従事．

入江 英嗣（正会員）

1975 年生．1999 年東京大学工学部電子情報工学科卒業．2004 年東京大学大学院情報理工学系研究科電子情報学専攻博士課程修了．博士（情報理工学）．同年より科学技術振興機構 CREST「ディペンダブル情報処理基盤」研究員．プロセッサアーキテクチャ等の研究に従事．

五島 正裕（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学大学院情報学研究科助手．2005 年東京大学情報理工学系研究科助教授，現在に至る．高性能計算機システムの研究に従事．博士（情報学）．2001 年情報処理学会山下記念研究賞，2002 年同学会論文賞受賞．IEEE 会員．

坂井　修一（正会員）

1981 年東京大学理学部情報科学科卒業．1986 年東京大学大学院工学系研究科修了，工学博士．電子技術総合研究所，MIT，RWC，筑波大学を経て，1998 年東京大学助教授．2001 年より東京大学教授（大学院情報理工学系研究科）．計算機システムおよびその応用の研究に従事．特に並列処理アーキテクチャ，相互結合網，最適化コンパイラ，省電力アーキテクチャ，ディペンダブルアーキテクチャ等の研究を進めている．情報処理学会研究賞（1989 年），同論文賞（1991 年），IBM 科学賞（1991 年），市村学術賞（1995 年），IEEE Outstanding Paper Award（1995 年），Sun Distinguished Speaker Award（1997 年）等受賞．著書『論理回路入門』（培風館),『コンピュータアーキテクチャ』（コロナ社）等．情報処理学会（現在理事），電子情報通信学会，人工知能学会，ACM，IEEE 会員．