Vol.2015-ARC-216 No.38
2015/8/6

マルチコアプロセッサのための スレッド間共有データを考慮したキャッシュ機構

西村 秦[†1]　佐藤 雅之[†2]　江川 隆輔[†2]　小林 広明[†2]

概要：マルチコアプロセッサでは，ラストレベルキャッシュ (LLC) を複数のコアで共有している．このため，LLC 上には複数のスレッドで共有しているデータ (共有データ) が存在する．複数のスレッドからアクセスされる共有データは，単一のスレッドからしかアクセスされないデータ (私有データ) に比べ再利用性が高い．しかし，共有データと私有データの区別を行わないデータ管理では，私有データによる過剰なキャッシュ占有のため，共有データを十分に保存できずヒット率の低下を招く．そこで，本研究では，並列プログラムの実効性能向上を目的とし，複数のスレッドに共有されるデータとそれ以外のデータを LLC 上で個別に管理するキャッシュ機構を提案する．提案手法は，LLC 上のデータを共有データと私有データに分けて管理することで，再利用性の高い共有データを優先的に LLC に保持する．これにより，共有データのヒット率が向上し，並列プログラムの実効性能向上が期待できる．シミュレーションによる評価結果から，提案手法は，LRU 置換ポリシに基づくキャッシュ機構と比較して最大 1.70 倍，平均 1.13 倍の性能向上を可能にすることが明らかとなった．

1. はじめに

ムーアの法則 [1] に従う半導体の集積密度の向上により，1 つのチップに複数のコアを搭載することが可能になった．このため，プロセッサの演算性能は飛躍的に向上している．一方で，演算に用いられるデータを保存するメインメモリのデータ転送性能の向上率は，プロセッサの演算性能の向上率よりも小さい．このため，計算機システムにおいて，メインメモリのデータ転送速度がボトルネックとなっている．

この問題を解決するために, 近年のプロセッサにはキャッシュメモリ (キャッシュ) を搭載している．キャッシュはメインメモリよりも高速なデータ転送を可能とし，メインメモリへのアクセス時間を隠蔽することができる．また，容量とデータ転送速度の異なるキャッシュをプロセッサ内に複数搭載することで，ヒット率の向上とアクセス時間の短縮を両立し，システムの高性能化を実現している．

近年のマルチコアプロセッサにおいて，LLC は複数のコアで共有されている．このため，複数のコアで分担処理を行う並列プログラムを実行した際に，共有データが LLC に保存される．共有データは複数のコアからアクセスされるため，私有データに比べアクセスが集中すると考えられる．したがって，共有データを優先的に LLC に残しておくことでヒット率が増加し，実効性能が向上する可能性がある．これまでにも共有データを考慮した LLC の管理機構は提案されている [2] [3]．しかし，これらの提案では，将来のアクセスパターンを予測する必要があることや，コア数が増大した際にハードウェアオーバーヘッドが大きくなる問題がある．

本研究では，並列プログラムの実効性能向上を目的とし，共有データを保護するキャッシュ管理機構を提案する．本機構では，LLC において少ないハードウェアオーバーヘッドで共有データと私有データを判別し，別々に管理する．これにより，再利用性の低い私有データによって共有データが追い出されることを防ぎ，共有データへのヒット数が増加する．その結果，並列プログラムの実効性能向上が期待できる．

2. 共有データが性能に与える影響

本章では，共有データを保護する必要性について述べる．まず，予備実験を通して共有データのアクセス特性を調査する．そして，従来手法のコア間共有キャッシュにおけるデータ管理機構の問題点について述べる．

[†1] 現在，東北大学 大学院情報科学研究科
[†2] 現在，東北大学 サイバーサイエンスセンター

© 2015 Information Processing Society of Japan

表 1　予備実験の環境

ベンチマーク	速度向上	MPKI
is	1.33	1.03
mg	1.36	10.02
canneal	1.42	13.84
ocean.ncont	1.57	20.86
ocean.cont	1.57	21.24
ft	1.78	2.78
streamcluster	1.88	7.82

表 2　予備実験の環境

パラメータ	値
CPU	8 コア，2.66GHz
L1I キャッシュ	4-way，32KB/1 コア，4 サイクル
L1D キャッシュ	8-way，32KB/1 コア，4 サイクル
L2 キャッシュ	8-way，256kB/1 コア，9 サイクル
LLC	16-way，8MB/8 コア，35 サイクル
メインメモリ	175 サイクル

2.1　共有データのアクセス特性

2.1.1　共有データの平均再利用回数

共有データは複数のコアからアクセスされるため，単一のコアからしかアクセスされない私有データに比べて再利用性が高い可能性がある．そこで，共有データのアクセス特性を調査するため，予備実験を行った．予備実験では，LLC 上で再利用されたブロックに着目し，共有データと私有データについての 1 ブロックあたりの再利用回数 (平均再利用回数) を求める．

予備実験は Sniper シミュレータ [4] を用いた．実験環境を表 2 に示す．全てのキャッシュは LRU 置換ポリシによって管理され，ブロックサイズは 64 バイトとする．ベンチマークには並列プログラムである PARSEC [5] と SPLASH-2 [6]，Nas Parallel Benchmark [7] から，LLC のデータ管理ポリシを変更することによって速度向上が得られる可能性のあるものを選択して用いた．表 1 に，選択したベンチマークの一覧を示す．これらのベンチマークは，容量が 8MB の場合において MPKI(Miss Per Kilo Instructions) が 1 以上であり，かつ，LLC 容量を 8MB から 16MB に増加させた際に 10%以上の速度向上が得られたベンチマークである．MPKI は，1000 命令あたりのミス数を表しており，1 より小さいとミスが性能に与える影響は小さいことを意味する [8]．シミュレーションの実行区間は，並列処理を行うスレッドが生成されてから破棄されるまでとする．

予備実験の結果を図 1 に示す．図 1 の縦軸は私有データ (private) と共有データ (shared) それぞれの平均再利用回数を示す．横軸はベンチマークと相乗平均 (GM) を示す．図 1 から，全てのベンチマークにおいて，私有データの平均再利用回数よりも，共有データの回数が平均で 2 倍以上多いことがわかる．したがって，私有データよりも共有データの方が再利用性が高いことがわかる．

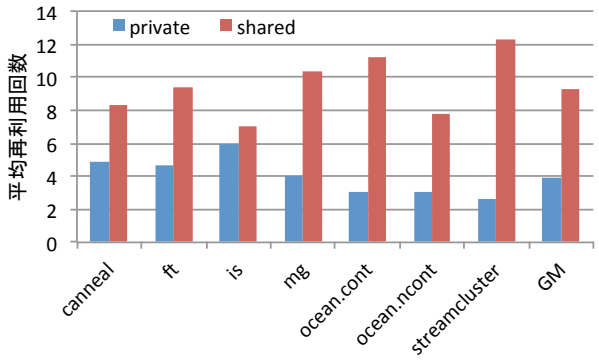

図 1　平均再利用回数

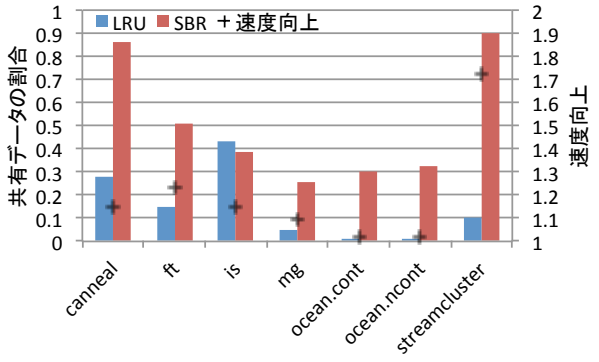

図 2　共有データの割合

2.1.2　共有データの LLC に占める割合

LLC において，共有データがどの程度の割合で保存されているかを確認するための予備実験を行った．実験環境は表 2 の通りとする．図 2 に予備実験の結果を示す．図 2 の左縦軸と棒グラフは LLC を占める共有データの割合を示し，横軸はベンチマークを示す．図 2 の LRU は LRU ポリシにおける共有データの割合を示す．図 2 より，全てのベンチマークで共有データは LLC の半分以下しか占めていないことがわかる．前節で示したとおり，共有データの平均再利用回数が多いにもかかわらず，そのキャッシュ中に占める割合は非常に低いといえる．

図 2 の SBR(Static Best Ratio) は，LLC の一部容量を共有データのために最適に割り当てた場合の，共有データの割合を示している．具体的には，共有データに割り当てるキャッシュ容量をウェイ数単位で 1 から 15 まで変化させて実験を行い，この中で最も高い性能を実現したウェイ数の結果を SBR として用いた．図 2 より，7 個のうち 6 個のベンチマークで SBR の共有データの割合は LRU ポリシの割合よりも高いことがわかる．このことは，LRU において共有データが十分に保存されず追い出されていることを示している．また，最大の性能が得られる共有データの割合は，ベンチマーク毎に異なっていることもわかる．

図 2 の右側縦軸とプロットは LRU に対して SBR で得ら

© 2015 Information Processing Society of Japan

れた性能向上を示す．図 2 から，全てのベンチマークにおいて，SBR は LRU と比較して性能が向上していることがわかる．is 以外のベンチマークでは，共有データの割合を増加させた結果として性能が向上しており，共有データを保護することで性能向上を達成できることが明らかとなった．一方で，is では共有データの割合を減少させることで性能が向上している．このことから，アクセス特性に応じて共有データの割合を適切に調整することが必要である．

2.2 関連研究

これまでにもデータの共有を考慮した管理ポリシは研究されている [2] [3]．これらの研究においても，共有データは再利用性が高いことが示されている [2]．また，ミスした際に複数のコアでストールを発生させるため，性能に与える影響が大きく，共有データを保護すべきであると述べられている [3]．

Natarajan ら [2] は，O_{one} と O_{all} の 2 つのキャッシュ置換ポリシを提案している．O_{one} は，2 つ以上のコアからアクセスされる共有データを保護するポリシである．また，O_{all} は，全てのコアからアクセスされる共有データを保護するポリシである．これらのポリシは将来においてどのデータがどのスレッドからアクセスされるかを予測する必要があるが，予測の正確性が低いことから現実的な実装には至っていない．

Muthukumar ら [3] は，置換によって追い出すデータを決定する際に，優先度を計算し，優先度が低いデータをキャッシュから追い出す Sharing and Hit based Prioritizing 置換ポリシ (SHP) を提案している．本ポリシにおける優先度は，データへの参照回数と，データへアクセスするスレッド数で表される共有度を用いて算出される．この優先度により，参照回数が多く，かつ，アクセスするスレッド数が多いデータはキャッシュから追い出されにくくなる．しかし，SHP では共有度を算出するために，アクセスしたコアを記録するビットをコア数分用意する必要がある．そのため，コア数の増加に伴ってハードウェアコストが増大するため，拡張性に乏しい．

異なるアクセス特性を持つデータを判別し，各々のデータ毎にキャッシュ資源を分割して管理を行う機構として，キャッシュ分割機構 [9] [10] が提案されている．Qureshi ら [9] は，Utility-based cache partitioning を提案している．スレッド毎にデータを判別し，ユーティリティモニタを用いて各スレッドが必要とするキャッシュ資源量を見積もることで，各スレッドに適切な容量の割当を行う．ユーティリティモニタは，再利用間隔とヒット数の関係を表すヒストグラム [11] に基づき，ヒット数がより増加するようにキャッシュ容量の割当を変化させる．

また，Khan ら [10] は Read-Write Partitioning(RWP) を提案している．RWP では，読み込みだけのデータ (Read データ) と書き込みがあるデータ (Write データ) を判別し，キャッシュの分割管理を行っている．Read データは，Write データよりも性能に影響を与えやすい．そのため，Read データと Write データを分割して管理することにより，互いに追い出し合うことを防ぐ．各データへの割当容量はユーティリティモニタを用いて決定される．

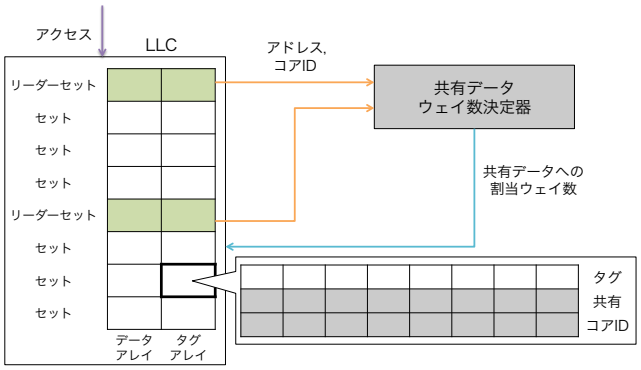

図 3 共有データを保護する機構の概要図

これらの動的キャッシュ分割機構では，独立した複数のプログラムを同時に実行するマルチプログラムワークロードを対象としており，データをそれぞれのプログラム毎に分割管理することを対象としている．そのため，並列プログラムを実行した際にキャッシュ上に生じる共有データについては考慮していない．したがって，LLC 上で共有データを判別することで，共有・私有データを LLC 上で分割管理することが可能になる．これにより，共有データの保護が可能になり，性能向上が期待できる．

3. 共有データを考慮した動的キャッシュ分割機構

共有データを保護するために，共有データと私有データをキャッシュ上で分割管理する機構を提案する．本機構の概要を図 3 に示す．本機構は，LLC のタグアレイの各ブロック毎に共有データを判別するための追加情報 (**共有ビット**，**コア *ID* フィールド**) と，共有データに割り当てるウェイ数を決定する**共有データウェイ数決定器**を必要とする．これらのハードウェアを用いて，セット上を共有データが占めるウェイ数と，共有データへの割当ウェイ数 (W_s) を把握し，データ置換時にセット上から追い出すデータを決定する．セット上の共有データ数は W_s 以下にならず，共有データの保護が可能になる．以下の節では，提案機構の詳細について述べる．

3.1 共有データへの割当ウェイ数の決定

共有データウェイ数決定器の概要を図 4 に示す．W_s を決定するために，セットサンプリング [12]，シャドウディレクトリ，ユーティリティモニタ [9] を用いる．セットサンプリングは，キャッシュアクセスのサンプリングを行

© 2015 Information Processing Society of Japan

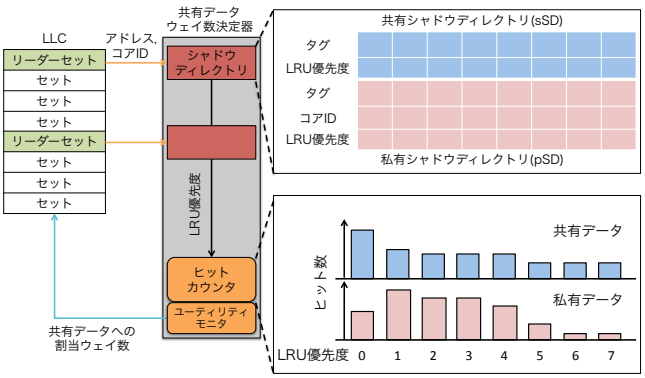

図 4 共有データウェイ数決定機構

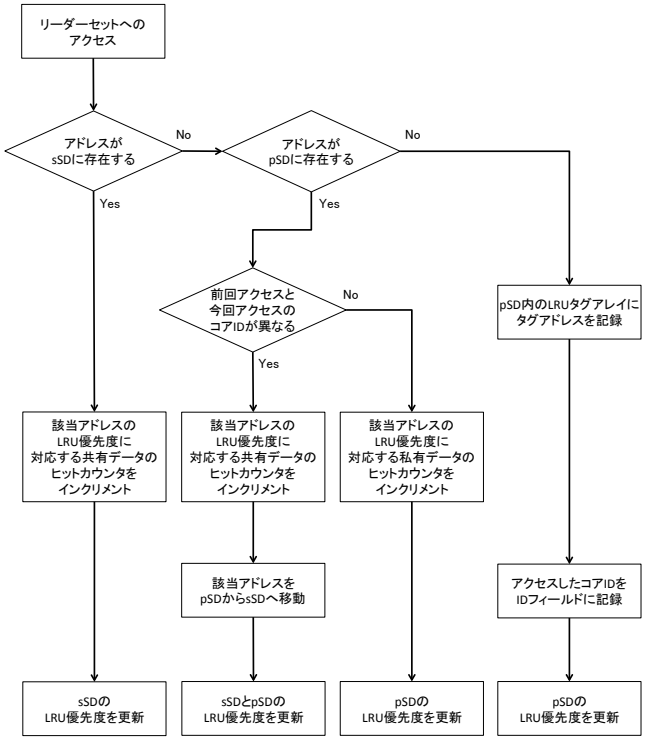

図 5 シャドウディレクトリ内の管理手順

う．サンプリングされたアクセスは，シャドウディレクトリを用いて共有データに対するヒットと私有データに対するヒットの判別を行い，それぞれのヒット数と再利用間隔 (LRU 優先度) の関係を明らかにする．この関係に基づき，ユーティリティモニタ [9] は，W_s を変化させた場合のヒットを予測し，W_s の増減を決定する．

3.1.1 セットサンプリングを用いたアクセス情報の取得

セットサンプリングは，キャッシュの一部のセットを用いてキャッシュへのアクセス情報を収拾する．サンプリングに用いられるセットのことをリーダーセットと呼ぶ．リーダーセットは LLC 上に 32 個 [10] あり，全セット数を N としたとき，$N/32$ 個の間隔で配置されている．これらのリーダーセットにアクセスが生じた場合，アドレスとアクセスしたコアの ID をシャドウディレクトリに通知する．

3.1.2 シャドウディレクトリを用いたヒット数計測

シャドウディレクトリには，共有データをサンプリングするためのシャドウディレクトリ (sSD) と私有データのためのシャドウディレクトリ (pSD) である．各ディレクトリは通常のタグアレイと同様に，ウェイ数分のタグアドレスと LRU 優先度を記録するためのビットを持つ．また，pSD についてはアクセスしたコアを記録しておくためのコア ID フィールドを持つ．

図 5 にシャドウディレクトリにおけるタグアレイの具体的な管理方法と共有データの判別方法を示す．アクセスが発生した際には，要求されたアドレスと同じアドレスが sSD に存在するかを確認する．sSD にアドレスが存在する場合，該当アドレスの LRU 優先度に対する共有ヒットがあったことをユーティリティモニタに通知する．そして，LRU ポリシに基づき sSD の LRU 優先度を更新する．

sSD に該当のアドレスが存在せず，pSD に存在する場合，今回アクセスしたコア ID と pSD に保存されている前回アクセスしたコア ID を比較する．もし，異なるコアからのアクセスであれば，そのデータは共有データであるとみなし，該当の LRU 優先度に対する共有ヒットがあったことをユーティリティモニタに通知する．その後，該当アドレスを sSD に保存するためにアドレスを LRU ポリシで置換し，pSD の該当アドレスを無効化する．一方で，前回のアクセスと同じコアからのアクセスであれば，そのデータを私有データであるとみなす．そして，該当アドレスの LRU 優先度に対応する私有ヒットがあったことをユーティリティモニタに通知する．その後，pSD を LRU ポリシに基づき更新する．

sSD と pSD のどちらにもアドレスが存在しなかった場合，pSD に新しいアドレスとコア ID を記録するために，LRU ポリシに基づきアドレスを置換する．

3.1.3 ユーティリティモニタによる割当ウェイ数の決定

ユーティリティモニタは，再利用間隔とヒット数の関係を評価するためのヒットカウンタを持つ．ヒットカウンタはキャッシュと同等のウェイ数分あり，共有データと私有データそれぞれのために 2 つ用意されている．ヒットカウンタは sSD または pSD でヒットが発生した場合，それぞれ共有データまたは私有データの通知された LRU 優先度に対応するヒットカウンタをインクリメントする．

ヒットカウンタの計測結果に基づき，ユーティリティモニタは W_s を変更した場合のヒット数を算出する．ユーティリティモニタの W_s 決定の具体例を図 6 に示す．図 6(a) は 8 ウェイのキャッシュにおけるヒット数の計測結果の例を示す．この計測結果から，任意の W_s におけるヒット数を予測することが可能である．図 6(a) では $W_s = 2$ であるため，共有データにおいては，LRU 優先度が 0 から 1 までのヒット数の合計が実際の LLC での共有データへのヒット数 (共有ヒット数) となる．また，私有データへの割当

© 2015 Information Processing Society of Japan

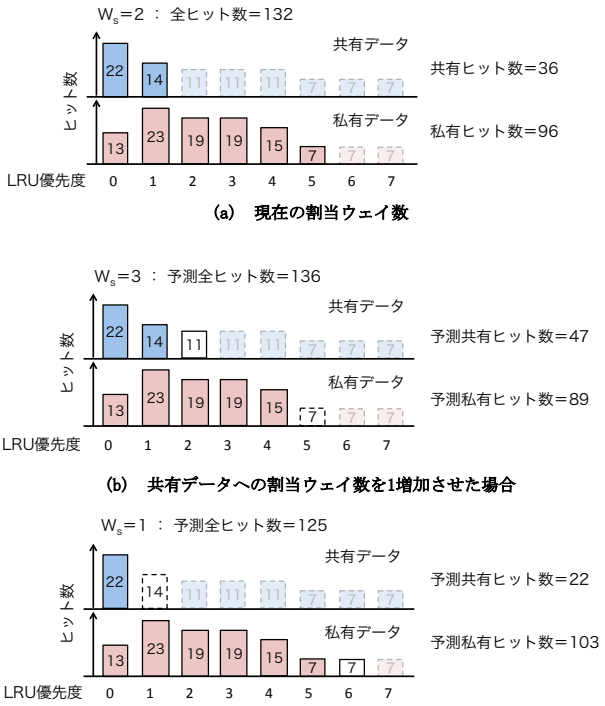

図 6 ヒットカウンタに基づくヒット予測

ウェイ数は 6 ウェイのため，LRU 優先度が 0 から 5 までのヒット数の合計が，実際の LLC での私有データへのヒット数 (私有ヒット数) となる．よって，共有ヒット数と私有ヒット数の和は実際の LLC での全ヒット数となる．

この計測結果を用い，ユーティリティモニタは W_s を変更した場合のヒット数を予測する．図 6(b) は図 6(a) の計測結果に基づき W_s を 1 増加させた場合を示す．W_s を 1 増加することにより，全ヒット数が図 6(a) の場合より増加することが予測される．一方で，図 6(c) は W_s を 1 減少させた場合を示す．W_s を 1 減少することにより，全ヒット数が図 6(a) の場合より減少することがわかる．

図 6(b) と図 6(c) の予測結果から，W_s を 1 増加することにより，ヒット数が増加することが予測されたので，ユーティリティモニタは W_s を 3 にする．もし，双方でヒット数の増加が予測された場合，よりヒット数の増加が望める方へ W_s を増加または減少させる．同じ増加量の場合，共有データに多く割り当てるように，W_s を 1 増加させる．

これらの W_s の更新は 5000 万サイクルごとに行われる．また，W_s の更新後は，すべてのヒットカウンタを 1 ビット右シフトする．これは，前の区間のアクセス傾向を考慮しつつ，現区間における最適なウェイ数を判定するためである．

以上の管理方法により，共有・私有データ毎に再利用間隔とヒット数の関係を評価することができる．

3.2 割当ウェイ数に基づく共有データの保護

3.2.1 LLC における共有データの判別方法

LLC では複数のコアからアクセスされる共有データと，単一のコアからのみアクセスされる私有データを判別する．これは，リーダーセット以外のセットでは，LLC のみで共有データと私有データを識別する必要があるためである．具体的には，データがキャッシュに挿入されてから追い出されるまでに，複数のスレッドにアクセスされた場合，本機構はそのデータを共有データと判別する．この判別のために LLC では，共有データを判別する共有ビットに 1 ビット，キャッシュブロック 1 つ当たりにアクセスしたコア ID を記録するコア ID フィールドを，n コアの場合 $\log n$ ビット分追加する．共有ビットは 0 で初期化され，コア ID フィールドは，置換時にデータにアクセスしたコアの ID を記録しておく．データにアクセスがあった際に，アクセスしたコア ID とフィールドに記録されているコア ID を比較し，異なれば，複数のスレッドからアクセスがあると判断し，共有ビットを 1 にする．一方で，同じであれば，ビットは 0 のままにする．

3.2.2 キャッシュ分割による共有データの保護

セット上の共有データ数と W_s を比較し，共有データ数が W_s を下回らないようにデータ置換を管理する．ここで，共有データの数を N_s とした際，追い出すデータを決定する方法を以下に示す．

- $N_s < W_s$
  共有データに与えられているウェイ数が少ない．このとき，次にデータが挿入される際には私有データを追い出す．
- $N_s > W_s$
  共有データに与えられているウェイ数が多い．このとき，次にデータが挿入される際には共有データを追い出す．
- $N_s = W_s$
  共有データに与えられているウェイ数は最適である．このとき，本機構は新たに挿入されるデータを必ず私有データと判別するので，共有データの割合を維持するために私有データを追い出す．

これにより，各セットにおいて共有データが必要とするウェイ数を割り当てることができる．

4. 性能評価

4.1 評価環境

提案手法の有効性を確認するために，LLC に提案手法を適用したプロセッサの評価を行う．評価は共有データの割合，MPKI，速度向上について，提案手法 (Proposal) と従来手法である LRU 置換ポリシ，SHP [3]，RWP [10]，SBR と比較を行う．評価環境は予備実験と同等とし，表 2 のパラメータを用いる．提案手法と RWP のベースポリシは

© 2015 Information Processing Society of Japan

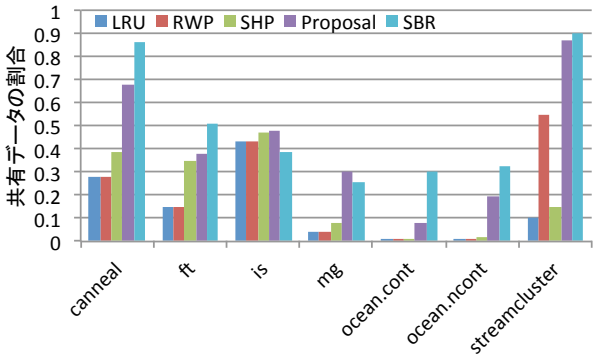

図 7 LLC を占める共有データの割合

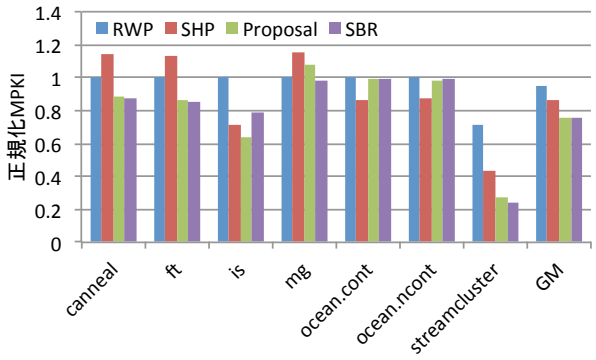

図 8 MPKI

LRU ポリシとする．ベンチマークは予備実験で選択したものを用いる．

4.2 共有データの割合の評価

提案手法により共有データが保護されているかを確認するために，共有データの割合を評価した．各ベンチマークの共有データの割合の結果を図 7 に示す．図 7 の縦軸は，各ポリシにおける共有データの割合を示し，横軸はベンチマークを示す．

図 7 から，提案手法の共有データの割合は，LRU の割合に比べ高いことがわかる．したがって，提案手法は共有データの保護を可能にすることが明らかとなった．また，各ベンチマークにおける提案手法の共有データの割合は，既存手法よりも高いことがわかる．したがって，共有データの保護は，既存手法よりも有効であることが明らかとなった．

また，提案手法の共有データの割合は，SBR と比較して少ないため，W_s を決定する機構に改善の余地があることがわかる．

4.3 MPKI の評価

提案手法により MPKI の削減が可能かを確認するために MPKI を評価した．各ベンチマークの MPKI の結果を図 8 に示す．図 8 の縦軸は，各ポリシにおける MPKI を

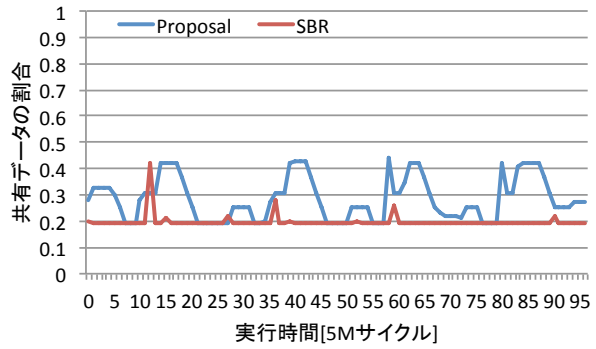

図 9 LLC を占める共有データの割合の時間変化：mg

示し，LRU ポリシの MPKI で正規化されている．横軸はベンチマークと相乗平均 (GM) を示す．図 8 から，提案手法は LRU ポリシと比較して，streamcluster で最大 73.0%，平均で 24.5%の MPKI を削減が得られることがわかる．また，RWP，SHP，と比較しても，それぞれ 19.9%，10.6%，の MPKI の削減が得られることがわかる．したがって，提案手法により並列プログラムの MPKI の削減が可能であることが明らかとなった．

streamcluster において提案手法が大幅に MPKI を削減した理由は以下の通りである．このベンチマークはストリームアクセスを行うプログラムである．ワーキングセットは 16MB [5] であるため，LRU ポリシで管理されている 8MB の LLC では私有データのスラッシングを引き起こす．このことは，実行区間における W_s は 15 を保ち続けたことからも明らかである．そして，このような場合セットに挿入される私有データは，次の置換によって直ちに追い出される．これにより，私有データのスラッシングから共有データを保護し，MPKI が減少した．

ocean.cont と ocean.ncont において，提案手法は MPKI を削減できていないが，SHP では MPKI を大幅に削減している．これは，これらのベンチマークの時間的局所性が低く，空間的局所性が高いためだと考えられる．また，キャッシュアクセスのほとんどが，私有データに対するアクセスであり，共有データを保護することで得られるヒット数は多くなかった．よって，LRU ポリシをベースとしている RWP と提案手法では MPKI を削減できず，アクセスが多いデータを保護する SHP は MPKI を大幅に削減したと考えられる．

mg において提案手法の MPKI は LRU ポリシに比べて増加していることがわかる．このベンチマークでは，提案手法により共有ヒット数は増加したが，私有ヒット数は大きく減少していた．この原因を解析するために，図 9 に mg の LLC を占める共有データの割合の時間変化を示す．図 9 の縦軸は 2 つのポリシにおける LLC を占める共有データの割合を示し，横軸は実行時間を示す．図 9 より，ほとんどのサンプリング区間において，提案手法の共有データの

© 2015 Information Processing Society of Japan

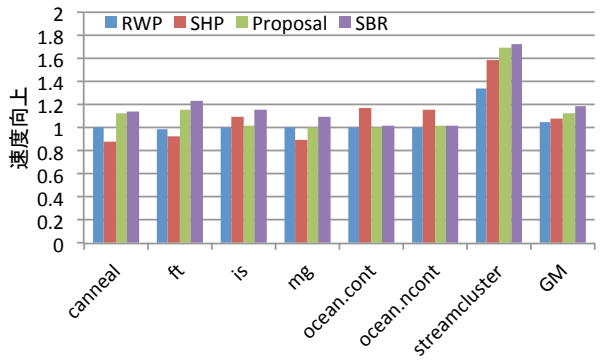

図 10 速度向上

割合は SBR の割合よりも高いことがわかる．よって，提案手法では共有データの過剰な保護が私有データのヒット数減少につながり，MPKI が増加したと考えられる．共有データの過剰な保護の原因究明は今後の課題である．

4.4 速度向上の評価

提案手法により性能向上が得られるかを確認するために，速度向上を評価した．各ベンチマークの速度向上の結果を図 10 に示す．図 10 の縦軸は，LRU ポリシに対する各ポリシの速度向上を示し，横軸はベンチマークと相乗平均を示す．図 10 から，提案手法は LRU ポリシと比較して最大 1.70 倍，平均で 1.13 倍の速度向上が得られることがわかる．また，RWP，SHP，と比較しても，それぞれ 1.08 倍，1.05 倍，の速度向上が得られることがわかる．したがって，本提案手法は並列プログラムの実効性能向上に有効であることが明らかとなった．

提案手法は，is の MPKI 削減に見合った性能向上が得られていない．これは，2.1 節で示したように，is は私有データが共有データによって追い出されているベンチマークであるためだと考えられる．しかし，提案手法が共有データの割合を増加させたため，私有データの割合が減少した．この結果，性能への影響が大きい私有データのヒット数が減少し，性能向上を得ることができなかったと考えられる．各データの性能に対する影響度の詳細な解析については今後の課題とする．

4.5 ハードウェアオーバーヘッドの評価

ハードウェアオーバーヘッドの比較評価を行う．まず，提案手法に必要な追加のハードウェアについて述べる．提案手法では，共有データを判別するためのコア情報を記録するビット，共有データと私有データの再利用間隔を計測するシャドウディレクトリ，ヒット数を計測するヒットカウンタの 3 つの追加のハードウェアが必要である．この 3 つについて，実験環境と同様のコア数 8，LLC の容量 8MB と仮定した場合の LRU ポリシに対するハードウェアオーバーヘッドについて以下に示す．

- 共有データの判別
  LLC の各ブロックには，共有データを判別するために 1 ビット，前回アクセスしたコアの ID を記録するためのビットを，コア数を n とした場合，$\log n$ ビット分必要である．したがって，64KB の追加のハードウェアが必要となる．
- シャドウディレクトリ
  セットサンプリングを行うシャドウディレクトリは，ウェイ数分のブロックが必要である．各ブロックには，タグアドレスを保存するために 27 ビット [13]，LRU 優先度を保存するために 4 ビット，コアのアクセス情報を記録するために 3 ビット必要である．また，共有データと私有データそれぞれのために 2 つのシャドウディレクトリが必要である，これらはサンプルセットの数だけ必要である．したがって，4.1KB の追加のハードウェアが必要となる．
- ヒットカウンタ
  分割サイズを動的に決定するためのヒットカウンタは 12 ビット [10] からなり，ウェイ数分必要である．また，ヒットカウンタは共有データと私有データのヒット数を計測するために 2 つ必要である．したがって，48B の追加のハードウェアが必要となる．

ユーティリティモニタに必要な比較器のハードウェアコストは非常に小さく無視できる [9] のでここでは考慮しない．以上をまとめると，本提案手法に必要な追加のハードウェアは 68.1KB となり，0.83%のハードウェアオーバーヘッドとなる．

次に，RWP のコストについて述べる．セットサンプリングに 4.2KB 必要であり，また，書き込みが行われたことを判別するために 1 ブロックあたり 1 ビット必要である．したがって，RWP に必要な追加のハードウェアは 20.2KB となり，0.25%のハードウェアオーバーヘッドとなる．

最後に，SHP のコストについて述べる．SHP は置換対象のブロックを決定するために優先度を計算する必要がある．優先度は，共有度とヒット数から計算されるため，LRU 優先度の情報を必要としない．共有度はデータにアクセスするコア数から算出されるため，コア数を n としたとき，1 ブロックあたり n ビット必要である．また，ヒット数を計測するために 4 ビット [14] 必要である．したがって，SHP に必要な追加のハードウェアは 128KB となり，1.56%のハードウェアオーバーヘッドとなる．

提案手法のハードウェアオーバーヘッドは RWP と比較すると 0.56%高いが，それに対する性能向上を考えると提案手法の方が優れている．また，提案手法のハードウェアオーバーヘッドは SHP を比較すると 0.73%低い．今回の評価環境では 8 コアで LLC 容量が 8MB を想定しているが，コア数を増大するとこのオーバーヘッドの差はさらに増大する．具体的に，64 コアで LLC 容量を 64MB を想定した

© 2015 Information Processing Society of Japan

場合，ハードウェアオーバーヘッドは SHP では 12.5%となるが，提案手法では 1.37%となる．したがって，提案手法は SHP よりも拡張性に優れていることがわかる．

5. おわりに

マルチコアプロセッサ上で並列プログラムを実行する場合，複数のスレッドで共有されるデータが LLC に保存される．しかし，再利用性の低い私有データの保存によって再利用性の高い共有データが追い出される問題がある．本論文では，並列プログラムの実効性能向上を目的とし，共有データを考慮したキャッシュ分割機構を提案した．本機構は，キャッシュ分割を用いて共有データのみを保存する領域を確保することにより，共有データを私有データから保護することが可能である．評価結果から，LRU 置換ポリシと比較して，最大で 1.70 倍，平均で 1.13 倍の性能向上が得られた．

今後の課題として，共有データの保護によって性能向上が得られないベンチマークの詳細な解析と，解析結果に基づく提案機構の改善が挙げられる．

参考文献

[1] Moore, G. E. et al.: Cramming more components onto integrated circuits, *Electronics Magazine*, p. 4 (1965).
[2] Natarajan, R. and Chaudhuri, M.: Characterizing multi-threaded applications for designing sharing-aware last-level cache replacement policies, *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, IEEE, pp. 1–10 (2013).
[3] Muthukumar, S. and Jawahar, P.: Sharing and Hit based Prioritizing Replacement Algorithm for Multi-Threaded Applications, *International Journal of Computer Applications*, Vol. 90, No. 12 (2014).
[4] Carlson, T. E., Heirman, W. and Eeckhout, L.: Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations, *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 52:1–52:12 (2011).
[5] Bienia, C., Kumar, S., Singh, J. P. and Li, K.: The PARSEC benchmark suite: characterization and architectural implications, *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, pp. 72–81 (2008).
[6] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P. and Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations, *ACM SIGARCH Computer Architecture News*, ACM, pp. 24–36 (1995).
[7] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S. et al.: The NAS parallel benchmarks, *International Journal of High Performance Computing Applications*, Vol. 5, No. 3, pp. 63–73 (1991).
[8] Rajan, K. and Govindarajan, R.: Emulating optimal replacement with a shepherd cache, *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, IEEE, pp. 445–454 (2007).
[9] Qureshi, M. K. and Patt, Y. N.: Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 423–432 (2006).
[10] Khan, S., Alameldeen, A. R., Wilkerson, C., Mutluy, O. and Jimenezz, D. A.: Improving cache performance using read-write partitioning, *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, IEEE, pp. 452–463 (2014).
[11] Chandra, D., Guo, F., Kim, S. and Solihin, Y.: Predicting inter-thread cache contention on a chip multi-processor architecture, *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, IEEE, pp. 340–351 (2005).
[12] Qureshi, M. K., Lynch, D. N., Mutlu, O. and Patt, Y. N.: A case for MLP-aware cache replacement, *ACM SIGARCH Computer Architecture News*, Vol. 34, No. 2, pp. 167–178 (2006).
[13] Intel: Intel Xeon Processor E7-8890 v3 (45M Cache, 2.50 GHz), `http://ark.intel.com/products/84685/Intel-Xeon-Processor-E7-8890-v3-45M-Cache-2_50-GHz`.
[14] Kharbutli, M. and Solihin, Y.: Counter-based cache replacement and bypassing algorithms, *Computers, IEEE Transactions on*, Vol. 57, No. 4, pp. 433–447 (2008).

© 2015 Information Processing Society of Japan