Vol.2012-SLDM-155 No.3
Vol.2012-EMB-24 No.3
2012/3/2

2コアプロセッサL1キャッシュ構成の正確で高速なシミュレーション手法

多和田 雅師[†1] 柳 澤 政 生[†2] 戸 川 望[†1]

近年，複数のコアをもつ組込みプロセッサが増えている．アプリケーションが限定される組込みシステムでは，速度や電力，面積の点で最適なキャッシュメモリが存在する．限定されたアプリケーションに対して複数のキャッシュ構成それぞれで動作シミュレーションを行うことで，キャッシュメモリ設計時に最適なキャッシュ構成を判定できる．マルチコアキャッシュ構成のシミュレーションは複雑になりシングルコアキャッシュ構成のシミュレーションよりも時間がかかってしまう．シングルコアプロセッサのキャッシュ構成ではシミュレーションの高速化手法が研究されているが，マルチコアプロセッサのキャッシュ構成ではシミュレーション高速化手法の研究は進んでいない．本稿では2コアプロセッサL1キャッシュのキャッシュ構成シミュレーションの高速化手法を提案する．マルチコアプロセッサではキャッシュコヒーレンシプロトコルがあり，複数の似たキャッシュ構成であっても内部状態が異なる場合が多い．そこでキャッシュコヒーレンシプロトコルの状態遷移とキャッシュ連想度に関する性質を利用することで1つのデータ構造で連想度の異なる複数のキャッシュ構成を表現する手法を提案する．複数のキャッシュ構成を1つのデータ構造で表し探索や更新の範囲を少なくすることで，シミュレーションの高速化を図る．

Fast and Exact Cache Configuration Simulation for Two-core L1 Cache

Masashi Tawada,[†1] Masao Yanagisawa[†2] and Nozomu Togawa[†1]

Recently, multiple-core processors are used in embedded systems very often. Since application programs running are much limited on embedded systems, there must exist an optimal cache memory in terms of power and area. Simulating application programs on various cache configurations is one of the best options to determine the optimal cache configuration. Multicore cache configuration simulation, however, is much more complicated and takes much more time than single-core cache configuration simulation. In this paper, we propose a very fast two-core L1 cache configuration simulation algorithm. We first propose a new data structure where just a single data structure represents two or more multicore cache configurations with different cache associativities. After that, we propose a new multicore cache configuration simulation algorithm using our new data structure associated with new theorems.

1. まえがき

近年のLSIの微細化に伴いキャッシュメモリの占める重要性は高くなっている．演算の処理速度に対し，プロセッサとメインメモリの通信速度が低くボトルネックとなる．キャッシュメモリを用いてメモリの階層化を行うことでこの速度差を緩和できる．組込みプロセッサでは特定アプリケーションのみが動作するため，特定アプリケーションの動作速度のキャッシュメモリへの依存度が高い．キャッシュメモリが小さすぎるとキャッシュミスが頻発し速度があまり向上しない．キャッシュメモリが大きすぎると速度は向上しても面積や電力のコストがかかる．アプリケーションに対しキャッシュメモリの構成を速度や電力，面積の点で最適化する必要がある．速度や電力を最適化するためにはアプリケーション動作時のキャッシュヒット/ミス回数を測定する必要がある．キャッシュヒット/ミス回数を測定する手法として，実際にシミュレーションしてキャッシュヒット/ミス回数を数える手法[2)–7),11),12),14)]とシミュレーションせずにキャッシュヒット/ミス回数を見積もる手法[1),9)]が存在する．前者は正確だが低速であり，後者は高速だが誤差が大きい．本稿では正確性に着目し，前者の手法を対象とする．シングルコアプロセッサの複数のキャッシュ構成を実際にシミュレーションする手法[14)]はすでに提案した．シングルコアプロセッサの複数のキャッシュ構成をそれぞれ動作シミュレーションするとき，既存の高速化手法として，複数のキャッシュ構成を1つのデータ構造にまとめる手法[7)]や，一部の探索から全体のキャッシュヒット/ミス回数を判定しシミューレーションを省略することで高速化する手法[14)]が存在する．一方，マルチコアプロセッサの1つのキャッシュ構成で動作シミュレーションする手法[15)]は存在するが，マルチコアプロセッサの複数のキャッシュ構成を対象にシミュレーションする手法は存在しない．

マルチコアプロセッサには各プロセッサのキャッシュ間で一貫性を保つために，キャッシュコヒーレンシプロトコルが存在する．キャッシュコヒーレンシプロトコルはキャッシュ内のデータに状態を結びつけて管理するプロトコルである．キャッシュコヒーレンシプロトコルにより，複数の似たキャッシュ構成であっても内部状態が異なる場合が存在する．マルチコアプロセッサの複数のキャッシュ構成を1つのデータ構造にまとめることは，キャッシュコ

†1 早稲田大学大学院基幹理工学研究科情報理工学専攻
Dept. of Computer Science and Engineering, Waseda University.
†2 早稲田大学大学院基幹理工学研究科電子光システム学専攻
Dept. of Electronic and Photonic Systems, Waseda University.

© 2012 Information Processing Society of Japan

ヒーレンシプロトコルが障害となる．マルチコアプロセッサで複数のキャッシュ構成を 1 つのデータ構造にまとめるには，異なるキャッシュ構成間の内部状態の違いをわずかなデータ付加で判定し，正しい状態へ復元するデータ構造とアルゴリズムが不可欠である．

本稿では 2 コアプロセッサ L1 キャッシュのキャッシュ構成シミュレーションの高速化手法を提案する．マルチコアプロセッサではキャッシュコヒーレンシプロトコルがあり，複数の似たキャッシュ構成であっても内部状態が異なる場合が多い．そこでキャッシュコヒーレンシプロトコルの状態遷移とキャッシュ連想度に関する性質を利用することで 1 つのデータ構造で連想度の異なる複数のキャッシュ構成を表現する手法を提案する．複数のキャッシュ構成を 1 つのデータ構造で表し探索や更新の範囲を少なくすることで，シミュレーションの高速化を図る．

2. キャッシュメモリ

キャッシュメモリはプロセッサとメインメモリの中間に位置し，データをバッファして速度を上げるメモリである．本稿ではキャッシュメモリの構成を定義する方式として，セットアソシアティブ方式を採用する．セットアソシアティブ方式はキャッシュメモリをセット数，ブロックサイズ，連想度の三つのパラメータで管理する．また，キャッシュ内のデータを追い出すアルゴリズムをキャッシュリプレースメントポリシと呼ぶ．本稿ではキャッシュリプレースメントポリシは LRU (Least Recently Used) とする．セット数はキャッシュを構成するセットの数である．セットはそれぞれがキャッシュリプレースメントポリシに沿って動作する優先度付きキューとみなせる．キャッシュ上で管理する情報の最小単位をブロックと呼ぶ．ブロックサイズはブロックの容量である．連想度はキャッシュを構成するセットが保持できる情報の数である．キャッシュメモリはセット数の数のセットから構成され，1 つのセットは連想度の数のブロックから構成される．LRU の優先度付きキューで各データの追い出し優先度を後に使われた順に $0, 1, 2, \ldots$ とする．セット数 s，ブロックサイズ b，連想度 a のキャッシュ構成 c を $c = (s, b, a)$ で表す．キャッシュ構成 (s, b, a) に対してメモリアクセスが発生したとき，アドレスはタグ，インデックス，オフセットに分割される．アドレスの下位 $\lg b$ ビットはオフセット，続く下位 $\lg s$ ビットはインデックス，残りのビットはタグとなる．図 1 にメモリアドレスのタグとインデックス，オフセットの分割を示す．タグはセット内のブロックにどのアドレスのデータが入っているかを示す．インデックスはどのセットに該当データが含まれるかを示す．オフセットはブロックの何バイト目が該当データかを示す．キャッシュ構成 c のインデックス i のセットを $S(c, i)$ で表す．セットはキャッシュリプレースメントポリシに沿って動作する優先度付きキューとみなせるため，セット内のブロックに優先度を定義できる．優先度が大きい順に追い出されるとする．セット $S(c, i)$ の優先度 j のブロックを $S(c, i)_j$ で表す．キャッシュ構成 $c = (16, 16, 4)$，メモリアクセス $A = 1010, 1010\,0000, 0000$ とするとタグは 1010, 1010，インデックスは 0000 である $S(c, 0000)$ と $S(c, 0000)_1$ の例は図 2 となる．メモリアクセス A はインデックス 0000 のセットの優先度 3 のブロックにデータが存在する．

3. キャッシュ構成シミュレーション

プログラムが動作するときプロセッサからメインメモリへのメモリアクセスはキャッシュメモリの存在を意識しない．あるアプリケーションが動作するときのメモリアクセスのリストを入手すれば，プログラムを再度実行することなくメモリアクセスをシミュレーションすることができる．このリストを**メモリアクセストレース**と呼ぶ．メモリアクセストレースとはメモリアドレスのシーケンスであり，各メモリアドレスはリード命令かライト命令かを付加情報として持つ．メモリアクセストレースを使い，特定の構成のキャッシュメモリでキャッシュヒット/ミス回数を数えるシミュレーションをキャッシュシミュレーションと呼ぶ．キャッシュ構成シミュレーションは複数のキャッシュ構成でキャッシュシミュレーションを行いキャッシュヒット/ミス回数を数えるシミュレーションである．

対象とするキャッシュ構成は
$s = s_0, 2s_0, 4s_0, \ldots, s_m$
$b = b_0, 2b_0, 4b_0, \ldots, b_m$
$a = 1, 2, 3, \ldots, a_m$
とする．ここで s_0，b_0 はセット数，ブロックサイズの最小値であり，s_m，b_m，a_m はセット数，ブロックサイズ，連想度の最大値である．

今，1 つのキャッシュ構成 $c = (s, b, a)$ を考える．キャッシュシミュレーションの全探索アルゴリズムを示す．

A1 キャッシュ構成 c に対しメモリアクセス A が発生したときインデックス i とタグ t を求める．インデックス i よりセット $S(c, i)$ を求める．

A2 セット $S(c, i)$ の優先度付きキューにタグ t が存在しているか判定する．

A3 もしステップ A2 でセット $S(c, i)$ に優先度 j でタグ t が存在していれば，メモリアクセス A はキャッシュ構成 c に対しキャッシュヒットとなる．またセ $S(c, i)_j$ の優先度ををキャッシュリプレースメントポリシに基づき更新する．

A4 もしステップ A2 でセット $S(c, i)$ にタグ t が存在していなければ，メモリアクセス A はキャッシュ構成 c に対しキャッシュミスとなる．またセット $S(c, i)$ にキャッシュリプレースメントポリシに基づきタグ t のブロックを追加する．

A5 メモリアクセスは存在するならステップ A1 へ行く．メモリアクセスが存在しないならば終了する．

キャッシュ構成シミュレーションはメモリアクセストレースに対し対象とする全てのキャッシュ構成のキャッシュヒット/ミス数を判定するシミュレーションである．

4. マルチコアプロセッサのキャッシュ構成シミュレーション

4.1 キャッシュコヒーレンシプロトコル

複数のプロセッサが同じメモリアドレスにアクセスする場合，データの整合性がとれなくなる可能性がある．これは各プロセッサ固有のキャッシュに最新でないデータが存在するために起こる．そのため，データの一貫性を保つための機構が必要となる．この一貫性をキャッシュコヒーレンシと呼ぶ．キャッシュコヒーレンシを保つためのプロトコルをキャッ

© 2012 Information Processing Society of Japan

Vol.2012-SLDM-155 No.3
Vol.2012-EMB-24 No.3
2012/3/2

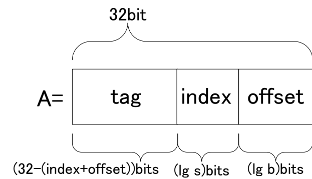

図 1 メモリアドレス 32bit，セット数 s，ブロックサイズ b，連想度 a の場合の tag, index, offset の各 bit 数.

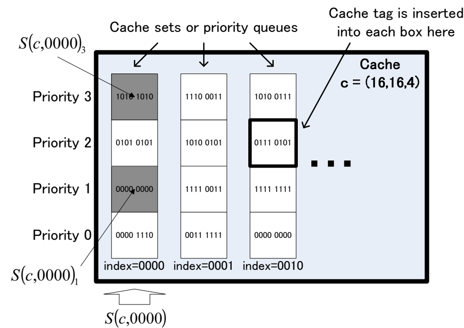

図 2 $S(c, 0000)$ と $S(c, 0000)_1$ の例.

シュコヒーレンシプロトコルと呼ぶ．キャッシュコヒーレンシプロトコルには，ライト・インバリデート型とライト・アップデート型がある[10)].

本稿ではキャッシュコヒーレンシプロトコルとして最も標準的なイリノイプロトコル[8)] を採用する．イリノイプロトコルはライト・インバリデート型プロトコルである．MESI プロトコルとも呼ばれる．キャッシュ上の各データは Modified, Exclusive, Shared, Invalid の 4 つの状態に遷移する．ライト・インバリデートはあるプロセッサからライト命令があったとき，他のプロセッサのキャッシュ上のデータを無効化することで一貫性を保つ．ライト・インバリデートはキャッシュヒット率が低いが，トラフィックが少なくすむ．表 1 に各状態が満たすべき性質を示す.

マルチコアプロセッサアーキテクチャでは他のプロセッサのデータアクセスを監視してキャッシュコヒーレンシを保つ．該当キャッシュのデータと同じデータが他のプロセッサに存在することをスヌープヒットと呼ぶ.

4.2 2 コアプロセッサのキャッシュ構成シミュレーション

本稿の対象アーキテクチャはキャッシュコヒーレンシプロトコルがイリノイプロトコル，

表 1 イリノイプロトコルの持つ状態の性質.

	Modified	Exclusive	Shared	Invalid
データの有効性	有効	有効	有効	無効
他のキャッシュとの関係性	排他	排他	共有	-
メインメモリとの整合性	変更有り	変更無し	変更無し	-

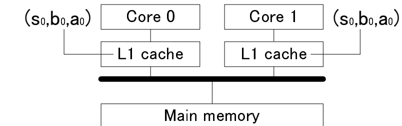

図 3 対象アーキテクチャ.

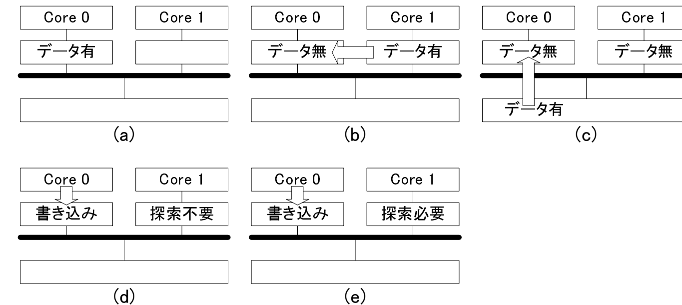

図 4 キャッシュ構成シミュレーションで回数を測定するべき状況.

キャッシュリプレースメントポリシが LRU の 2 コアプロセッサプライベート L1 キャッシュのアーキテクチャである．対象とするアーキテクチャを図 3 に示す．Core 0 と Core 1 の各プロセッサそれぞれのキャッシュメモリの構成は同じとする.

キャッシュシミュレーションの目的はキャッシュヒット/ミス回数を測定して消費エネルギーや速度を計算することである．マルチコアプロセッサのキャッシュシミュレーションでは，メモリアクセスの発生したプロセッサのキャッシュメモリのキャッシュヒット/ミス回数だけでなく，他のプロセッサのキャッシュメモリのスヌープヒット/ミス回数も含めて測定する必要がある．マルチコアプロセッサのキャッシュ構成シミュレーションで回数を数えるべき状況を以下に示す.

リード命令がキャッシュヒットした場合 Core0 でリード命令が発生し，このメモリアクセスが Core0 の L1 キャッシュでキャッシュヒットした場合を考える．該当データのキャッシュコヒーレンシプロトコルの状態に関わらず Core1 の L1 キャッシュを探索する必要がない．キャッシュメモリ，メインメモリ間でデータの通信は発生しな

© 2012 Information Processing Society of Japan

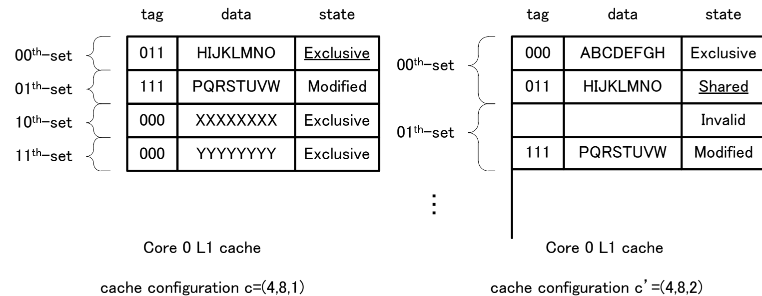

図 5 キャッシュメモリの内部状態．

い．図 4(a) にこの状況を示す．

リード命令がキャッシュミスした場合 Core0 でリード命令が発生し，このメモリアクセスが Core0 の L1 キャッシュでキャッシュミスした場合を考える．該当データが Core1 の L1 キャッシュに存在するかどうか調べるため Core1 のキャッシュを探索する必要がある．Core1 でスヌープヒットしたとき，キャッシュメモリ間でデータの通信が発生する．図 4(b) にこの状況を示す．Core1 でスヌープミスしたとき，キャッシュメモリとメインメモリ間でデータの通信が発生する．図 4(c) にこの状況を示す．

ライト命令がキャッシュヒットした場合 Core0 でライト命令が発生し，このメモリアクセスが Core0 の L1 キャッシュでキャッシュヒットした場合を考える．該当データのキャッシュコヒーレンシプロトコルの状態が Modified または Exclusive のとき，Core1 のキャッシュに同じデータは存在しないため，Core1 のキャッシュを探索する必要はない．図 4(d) にこの状況を示す．該当データのキャッシュコヒーレンシプロトコルの状態が Shared のとき，Core1 のキャッシュに同じデータは存在する可能性があるため，Core1 のキャッシュを探索する必要がある．図 4(e) にこの状況を示す．

ライト命令がキャッシュミスした場合 Core0 でライト命令が発生し，このメモリアクセスが Core0 の L1 キャッシュでキャッシュミスした場合を考える．該当データのキャッシュコヒーレンシプロトコルの状態に関わらず Core1 の L1 キャッシュを探索する必要がある．図 4(e) にこの状況を示す．

マルチコアプロセッサのメモリアクセストレースとはメモリアドレスのシーケンスであり，各メモリアドレスはどのプロセッサからのアクセス命令か，リード命令かライト命令かを付加情報として持つ．

キャッシュ構成を変えながら図 4(a),(b),(c),(d),(e) の 5 つの状況が発生した回数をそれぞれ数えることで，各キャッシュ構成でのキャッシュメモリ間でのデータの通信回数，メインメモリとキャッシュメモリでのデータの通信回数がわかる．データの通信回数からキャッシュメモリ動作時の遅延時間や消費エネルギーを計算できる．

5. 高速化アルゴリズムの提案

キャッシュ構成シミュレーションは，単一構成のキャッシュシミュレーションを複数回行うことで実現できる．しかし，この手法は現実的でない時間がかかる可能性がある．複数のキャッシュ構成をまとめて同時にシミュレーションすることができれば実行時間を短縮できる．複数のキャッシュ構成をまとめて同時にシミュレーションするためには，同時に複数のキャッシュ構成を表現するデータ構造が必要となる．ひとつのデータ構造を探索，更新することで複数のキャッシュ構成で探索，更新が行われるようなデータ構造を構築することができれば高速なキャッシュ構成シミュレーションを実現できる可能性がある．本稿では連想度に着目し，連想度の異なる複数のキャッシュ構成をひとつのデータ構造で表現する手法を提案する．

図 3 の 2 コアプロセッサの Core 0 と Core 1 のキャッシュについてセット数，ブロックサイズが等しく，連想度の異なる 2 つのキャッシュ構成 $c = (4, 8, 1)$ と $c' = (4, 8, 2)$ を考える．これらのキャッシュ構成を対象にメモリアクセストレースを与えれば，キャッシュ構成 c と c' では同じインデックスのセットを比べたとき，連想度が小さい構成 c に属するデータは連想度が大きい構成 c' のに属するデータにすべて含まれる．図 5 に c と c' のそれぞれの Core 0 のキャッシュの内部状態を示す．c のインデックス 00 のセットのタグ 011 のブロックのデータは HIJKLMNO であり，c' のインデックス 00 のタグ 011 のブロックのデータも HIJKLMNO である．異なるキャッシュ構成間で同じインデックスのセットかつ同じタグならばデータは一致する．一方，c のインデックス 00 のセットのタグ 011 のブロックの状態は Exclusive であり，c' のインデックス 00 のタグ 011 のブロックの状態は Shared である．キャッシュコヒーレンシプロトコルにより決定される状態は異なることがある．

まとめると，セットを優先度付きキューとみなしたときの各ブロックの優先度は一致する．セットを優先度付きキューとみなしたときの各ブロックのデータと優先度は一致するが，キャッシュコヒーレンシプロトコルにより決定される状態は異なる．

c と c' をひとつのデータ構造で表現しよう．c' のデータ構造のみを保持し c と c' をひとつのデータ構造で表現すると，c と c' それぞれのキャッシュコヒーレンシプロトコルの状態を表現できず，キャッシュコヒーレンシプロトコルの状態は異なるとき間違ったシミュレーションをしてしまう．提案データ構造を図 6 に示す．複数のキャッシュ構成に対して同じメモリアクセスが発生してもキャッシュ内のキャッシュコヒーレンシプロトコルの状態が Exclusive と Shared で異なることが問題となる．そこで新たなデータ構造を導入して似たキャッシュ構成を同時に表すことを考える．問題を解決するために Exclusive と Shared の状態のどちらかを表す状態 ES を導入する．また，同じキャッシュ構成の別のキャッシュに同じデータが存在する場合にどの位置にデータが存在するかをポインタで示す．ポインタだけだとそのデータの状態が一見 Exclusive でも実際は Shared の場合が存在するため，判定できるようにデータごとに判定値をつける．表現したいキャッシュ構成の連想度が該当データの判定値よりも大きい場合はそのデータは Shared, 判定値以下の場合は Exclusive とする．

キャッシュコヒーレンシプロトコルにより決定される状態をデータ構造に情報を付加して表すことで，複数のキャッシュ構成をひとつのデータ構造で表現する必要がある．

そこで，c' のデータ構造に状態を表現するためのポインタと判定値を付加したデータ構造を提案する．すなわち，Exclusive と Shared を同時に表す状態 ES, 及び他のプロセッサのキャッシュと同じデータを結びつけるポインタを提案する．

さらにこのデータ構造を用いてキャッシュメモリの状態を判定，更新するアルゴリズムを

© 2012 Information Processing Society of Japan

Vol.2012-SLDM-155 No.3
Vol.2012-EMB-24 No.3
2012/3/2

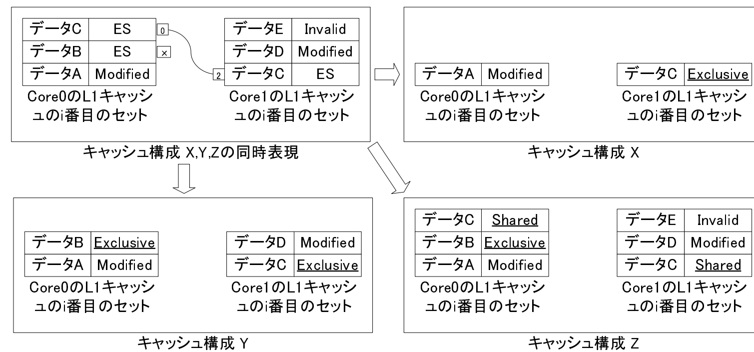

図 6 提案データ構造.

提案する．セット数 s，ブロックサイズ b，連想度 $a = 1, 2, 3, \ldots, a_m$ のキャッシュ構成に対して，Core 0 のインデックス i のセットにタグ t となるメモリアクセス A があったとする．

リード命令アルゴリズム

R1 Core 0 のインデックス i のセットを表現するデータ構造で対しタグ t を持つブロックを探索する．優先度 p に該当するブロックに存在するときステップ R2 へ行く．セット内に該当ブロックが存在しないときステップ R10 へ行く．

R2 連想度 a が $a > p$ となるキャッシュ構成でキャッシュヒットする．$a \leq p$ となるキャッシュ構成でキャッシュミスする．

R3 該当ブロックの状態が Modified ならばステップ R4 へ行く．状態が ES ならばステップ R5 へ行く．

R4 連想度 a が $a > p$ となるキャッシュ構成で図 4(a) の状況となる．$a \leq p$ となるキャッシュ構成で図 4(c) の状況となる. ステップ R14 へ行く.

R5 該当ブロックにポインタが存在するときステップ R6 へ, 存在しないときステップ R8 へ行く.

R6 ポインタ先の優先度を p' とする．連想度 a が $a > p$ となるキャッシュ構成で図 4(a) の状況となる．$a \leq p$ かつ $a > p'$ となるキャッシュ構成で図 4(b) の状況となる．$a \leq p$ かつ $a \leq p'$ となるキャッシュ構成で図 4(c) の状況となる.

R7 該当データの判定値 j と p を比較し,$j < p$ ならば判定値 j に p を代入する. ポインタの先の優先度 p' と比較し,$j > p'$ ならば判定値 j に p' を代入する. ポインタの先のデータの判定値を 0 にする．ステップ R14 へ行く.

R8 連想度 a が $a > p$ となるキャッシュ構成で図 4(a) の状況となる．$a \leq p$ となるキャッシュ構成で図 4(c) の状況となる.

R9 判定値 j と p を比較し,$j < p$ ならば判定値 j に p を代入する．ステップ R14 へ行く.

R10 Core 1 のインデックス i のセットを表現するデータ構造で対しタグ t を持つブロックを探索する．優先度 p' で存在するときステップ R11 へ行く．セット内に該当ブロックが存在しないときステップ R13 へ行く.

R11 連想度 a が $a > p'$ となるキャッシュ構成で図 4(b) の状況となる．$a \leq p'$ となるキャッシュ構成で図 4(c) の状況となる.

R12 状態 ES，判定値 p' で A のデータを含むブロックを追加する．Core 0 の該当ブロックと Core 1 の該当ブロックをそれぞれポインタでつなげる．と Core 1 の該当ブロックの判定値を 0 にする．ステップ R14 へ行く.

R13 すべての連想度のキャッシュ構成で図 4(c) の状況となる．状態 ES，判定値 a_m で A のデータを含むブロックを追加する.

R14 キャッシュリプレースメントポリシに従い優先度を更新する．終了する.

ライト命令アルゴリズム

W1 Core 0 のインデックス i のセットを表現するデータ構造で対しタグ t を持つブロックを探索する．優先度 p に該当するブロックに存在するときステップ W2 へ行く．セット内に該当ブロックが存在しないときステップ W9 へ行く.

W2 該当ブロックの状態が Modified ならばステップ W3 へ行く．状態が ES ならばステップ W4 へ行く.

W3 すべての連想度のキャッシュ構成で図 4(d) の状況となる．ステップ W10 へ行く.

W4 該当ブロックにポインタが存在するときステップ W5 へ, 存在しないときステップ W7 へ行く.

W5 連想度 a が $a > p$ となるキャッシュ構成で図 4(e) の状況となる．$a \leq p$ となるキャッシュ構成で図 4(d) の状況となる.

W6 ポインタ先のブロックの状態を Invalid にする．該当ブロックの状態を Modified にする．ステップ W10 へ行く.

W7 連想度 a が $a > p$ となるキャッシュ構成で図 4(e) の状況となる．$a \leq p$ となるキャッシュ構成で図 4(d) の状況となる.

W8 該当ブロックの状態を Modified にする．ステップ W10 へ行く.

W9 すべての連想度のキャッシュ構成で図 4(e) の状況となる．状態 Modified で A のデータを含むブロックを追加する.

W10 キャッシュリプレースメントポリシに従い優先度を更新する．終了する.

提案したデータ構造に対し，以下の定理が成立する.

定理 1. 状態 *Modified* または *ES* を持つ優先度 p のブロックでリード命令またはライト命令がキャッシュヒットしたときこのデータ構造は保たれる.

(証明略)

定理 2. あるセットでリード命令またはライト命令がキャッシュミスしたときこのデータ構造は保たれる.

(証明略)

提案したデータ構造を使い, 複数のキャッシュ構成の内容を同時に表現することでシミュレーションを高速化する手法を提案する．セット数, ブロックサイズが固定で異なる連想度をもつキャッシュ構成での同時シミュレーションが可能である.

提案アルゴリズム

P1 セット数 s，ブロックサイズ b を s_0，b_0 に初期化する.

P2 メモリアクセス A が発生, セット数 s, ブロックサイズ b からタグ t, インデックス i を計算する.

P3 メモリアクセス A がリード命令のとき，ステップ R1 へ行く，メモリアクセス A がライト命令のとき，ステップ W1 へ行く.

P4 **リード命令アルゴリズム**，または**ライト命令アルゴリズム**終了後，s に $s+1$ を代入する．$s \leq s_m$ ならばステップ P2 へ行く.

P5 s に s_0 を代入する．b に b_0 を代入する．$b \leq b_m$ ならばステップ P2 へ行く.

P6 メモリアクセスが存在すればステップ P1 へ行く.

前述したアルゴリズムには CRCB 手法[14] を適用できる[13].

6. 提案手法の評価

提案手法を C 言語で実装した．全探索する手法と提案手法の実行時間を比較した．使

© 2012 Information Processing Society of Japan

表 2　全探索手法と提案手法の実行時間の比較.

	全探索手法 [sec]	提案手法 [sec]
FFT	96.01 (1)	17.15 (0.18)
Speech	346.90 (1)	64.62 (0.19)

表 3　全探索手法と提案手法の出力，図 4(a),(b),(c),(d),(e) の 5 つの状況が発生した回数.

		全探索手法					提案手法				
		(a)	(b)	(c)	(d)	(e)	(a)	(b)	(c)	(d)	(e)
FFT	(8,8,1)	175343	48277	2714903	59096	1329635	175343	48277	2714903	59096	1329635
	(16,16,4)	858849	199137	1880537	328474	1060257	858849	199137	1880537	328474	1060257
	(32,32,16)	1850405	300963	787155	635702	753029	1850405	300963	787155	635702	753029
Simple	(8,16,2)	1588162	243963	9714057	394665	3492847	1588162	243963	9714057	394665	3492847
	(16,32,4)	1952807	305588	9287787	529772	3357740	1952807	305588	9287787	529772	3357740
	(32,8,8)	2085191	386515	9074476	658930	3228582	2085191	386515	9074476	658930	3228582

用した計算機はプロセッサが AMD 1.3GHz であり，メインメモリが 16GB の PC である．探索対象とするキャッシュ構成はセット数が 8 から 32, ブロックサイズが 8Byte から 32Byte, 連想度が 1 から 16 の計 45 構成である．Windows 用 GUI キャッシュシミュレータの SMPCache のベンチマークアプリケーションとして 2 コア FFT，Simple のアクセストレースを入力とする．各構成でのエネルギーと遅延速度を計算するための状況数を出力とする．全探索手法と提案手法の実行時間の比較を表 2 に示す．全探索手法と提案手法の出力である図 4(a),(b),(c),(d),(e) の 5 つの状況が発生した回数の一部を表 3 に示す．表 3 より全探索手法と提案手法で図 4(a),(b),(c),(d),(e) の 5 つの状況が発生した回数が一致することがわかる．提案手法は従来の手法に比べ約 5 倍高速化となった.

7. おわりに

本稿ではマルチコアプロセッサキャッシュのシミュレーションを高速化した．特に 2 コアプロセッサ L1 キャッシュについてキャッシュコヒーレンシプロトコルの動作を考察した．2 コアプロセッサ L1 キャッシュのキャッシュ構成シミュレーション高速化手法を提案し，計算機上で実装し，評価した．提案手法は従来の手法に比べ約 5 倍高速化となった.

謝　辞

本研究は一部 NEC との NEDO「ノーマリーオフコンピューティング基盤技術開発」共同研究業務による.

参 考 文 献

1) W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria, “A design framework to efficiently explore energy-delay tradeoffs,” in *Proc. CODES 2001,* 2001, pp. 260–265.
2) M. S. Haque, A. Janapsatya, and S. Parameswaran, “SuSeSim: a fast simulation strategy to find optimal L1 cache configuration for embedded systems,” in *Proc. CODES+ISSS 2009,* 2009, pp. 295–304.
3) M. S. Haque, J. Peddersen, A. Janapsatya, and S. Parameswaran, “DEW: a fast level 1 cache simulation approach for embedded processors with FIFO replacement policy,” in *Proc. DATE 2010,* 2010, pp. 496–501.
4) M. S. Haque, J. Peddersen, A. Janapsatya, and S. Parameswaran, “SCUD: A fast single-pass L1 cache simulation approach for embedded processors with round-robin replacement policy,” in *Proc. DAC 2010,* 2010, pp. 356–361.
5) M. S. Haque, J. Peddersen, and S. Parameswaran, “CIPARSim: Cache intersection property assisted rapid single-pass FIFO cache simulation technique,” in *Proc. ICCAD 2011,* 2011, pp. 126–133.
6) M. D. Hill and A. J. Smith, “Evaluating associativity in CPU caches,” *IEEE Trans. Computers*, vol. 38, no. 12, pp. 1612–1630, 1989.
7) A. Janapsatya, A. lgnjatovic, and S.Parameswaran, “Finding optimal L1 cache configuration for embedded systems,” in *Proc. ASP-DAC 2006,* 2006, pp. 796–801.
8) M. S. Papamarcos and J. H. Patel, “A low-overhead coherence solution for multiprocessors with private cache memories,” in *Proc. ISCA 84,*1984, pp. 348–354.
9) J. J. Pieper, A. Mellan, J. M. Paul, D. E. Thomas, and F. Karim, “High level cache simulation for heterogeneous multiprocessors,” in *Proc. DAC 2004,* 2004, pp. 287–292.
10) P. Stenstrom, “A survey of cache coherence schemes for multiprocessors,” *Computer,* vol. 23, no. 6, pp. 12–24,1990.
11) R. A. Sugumar “Set-associative cache simulation using generalized binomial trees,” *ACM Trans. Computer Systems*, vol. 13, No.1, pp. 32–56, 1995.
12) M. Tawada, M. Yanagisawa, T. Ohtsuki, and N. Togawa, “Exact, Fast and Flexible L1 Cache Configuration Simulation for Embedded Systems”, *IPSJ Transactions on System LSI Design Methodology*, vol.4, pp. 166–181, 2011.
13) 多和田雅師, 柳澤政生, 戸川望, “2 コアプロセッサアーキテクチャを対象とする正確なキャッシュ構成シミュレーションの高速化に対する一考察,” 2011 年電子情報通信学会ソサイエティ大会, A–3–11, 2011.
14) N. Tojo, N. Togawa, M. Yanagisawa, and T. Ohtsuki, “Exact and fast L1 cache simulation for embedded systems,” in *Proc. ASP-DAC 2009,* 2009, pp. 817–822.
15) M. Vega, J. Sanchez, R. Montafia, and F. Zarallo, “Simulation of cache memory systems on symmetric multiprocessors with educational purposes,” in *Proc. I International Congress in Quality and in Technical Education Inovation,* 2000, pp. 47–59.

© 2012 Information Processing Society of Japan