Vol.2013-SLDM-160 No.15
Vol.2013-EMB-28 No.15
2013/3/13

2コアアーキテクチャを対象とするトレースベースキャッシュシミュレーションの精度評価

多和田 雅師[1]　柳澤 政生[2]　戸川 望[1]

概要：一般にプロセッサ上でアプリケーションを走らせた場合にキャッシュがどのように動作するかサイクル精度でシミュレーションすると時間がかかる．そこで，特定のキャッシュ構成を想定してサイクル精度でシミュレーションすることによりメモリアクセストレースを入手し，メモリアクセストレースを用いてキャッシュ動作をトレースベースシミュレーションするとシミュレーション時間を極めて短くできる．ここでキャッシュのトレースベースシミュレーションとは，メモリアクセストレースに従ってプロセッサがメモリアクセスすると仮定し，キャッシュがどのように動作するかのシミュレーションである．ところが，マルチコアアーキテクチャではメモリアクセスは原理的に，想定するキャッシュ構成によって変化する．トレースベースシミュレーションをマルチコアアーキテクチャに適用した場合，メモリアクセストレースを入手するときに想定したキャッシュ構成とトレースベースシミュレーションで想定したキャッシュ構成が異なるとトレースベースシミュレーション結果はサイクル精度シミュレーション結果と一致しない．本稿では，メモリアクセストレースを入手するときに想定したキャッシュ構成とトレースベースシミュレーションで想定したキャッシュ構成が異なるとき，トレースベースシミュレーションがどの程度，サイクル精度シミュレーションと一致するかを評価する．

キーワード：キャッシュメモリ, キャッシュシミュレーション, トレースベースシミュレーション

Accuracy Evaluation of Trace-based Cache Simulation for Two-core L1 Caches

Tawada Masashi[1]　Yanagisawa Masao[2]　Togawa Nozomu[1]

Abstract: In trace-based cache simulation, we perform cache simulation based on a particular *memory access trace* obtained by cycle-accurate memory simulation. While cycle-accurate simulation takes too many time to run, trace-based cache simulation runs very fast and then we can evaluate many cache configurations in a short time. Let us consider a multi-core processor cache. We can obtain a memory access trace by using a cycle-accurate memory simulation but it can be changed when we consider another multi-core processor cache configuration. One of the main concerns in trace-based cache simulation applied to multi-core processor caches is its accuracy when the cache configuration that the memory access trace assumed is different from those the trace-based cache simulation targets. In this paper, we evaluate how much memory access traces affect cache configuration simulation when cache configurations simulated are different from the one that memory access traces assume, using several benchmark applications.

Keywords: cache memory, cache simulation, trace-based simulation

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

[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.

Vol.2013-SLDM-160 No.15
Vol.2013-EMB-28 No.15
2013/3/13

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], [3], [4], [5], [6], [7], [8], [13], [14], [15] 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], [11] 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 [15] はすでに存在する．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 [15] が存在する．一方，マルチコアプロセッサのキャッシュ構成で動作シミュレーションする手法 [16] は存在するが，コヒーレンシを考慮するかどうかが問題となる．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．

マルチコアアーキテクチャでは原理的にメモリアクセスは想定するキャッシュ構成によって変化する．そのため，トレースベースシミュレーションはサイクル精度シミュレーションより高速に動作するが，マルチコアアーキテクチャではメモリアクセストレースを入手するときに想定したキャッシュ構成とトレースベースシミュレーションで想定したキャッシュ構成が異なるときのトレースベースシミュレーションの結果はサイクル精度シミュレーションの結果と一致しないという問題が存在する．本稿では，メモリアクセストレースを入手するときに想定したキャッシュ構成とトレースベースシミュレーションで想定したキャッシュ構成が異なるときのトレースベースシミュレーションの精度を評価する．

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．

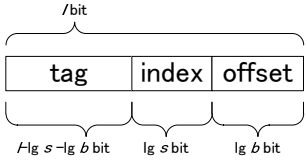

図 1 メモリアドレス 32bit，セット数 s，ブロックサイズ b，連想度 a の場合の tag, index, offset の各 bit 数．

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．

Vol.2013-SLDM-160 No.15
Vol.2013-EMB-28 No.15
2013/3/13

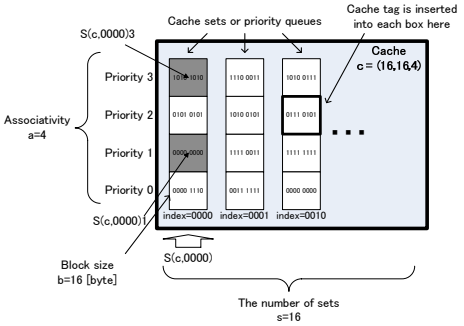

図 2 $S(c, 0000)$ と $S(c, 0000)_1$ の例.

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 [12].

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 MESI プロトコル [9] を採用する．MESI 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．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．MESI プロトコルでは他のプロセッサのキャッシュと整合性がとれていてメインメモリとの整合性がとれていない状態は存在しない．ライト命令により変更があったデータは，他のプロセッサでリード命令があったとき必ずメインメモリと整合性をとってから他のプロセッサのキャッシュに入ることになる．図 3 に MESI プロトコルの状態遷移図を示す．

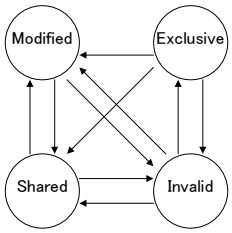

図 3 MESI プロトコルの状態遷移図.

表 1 MESI 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	-

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

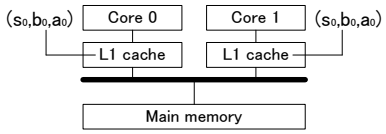

図 4 対象アーキテクチャ．

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MESIプロトコル, 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4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．このキャッシュアーキテクチャの構成を $(s_0, b_0, a_0)^2$ と表記する．

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．

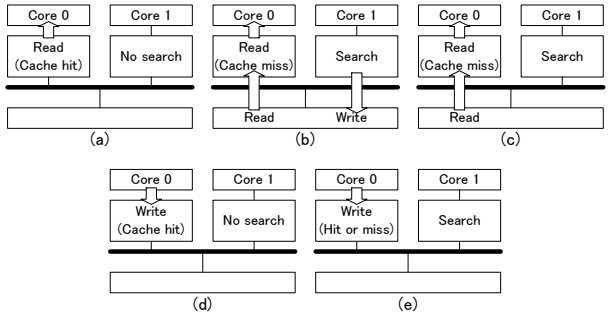

図 5 キャッシュ構成シミュレーションで回数を測定するべき状況．

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い．図5(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，キャッシュメモリからメインメモリへデータの書き戻しが発生し，その後該当キャッシュへのデータの通信が発生する．図5(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5(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5(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5(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5(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5(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のデータの読み書きの回数，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．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する手がかりとなる．

5. アクセストレース収得環境による動作の違い

シングルプロセッサアーキテクチャに対するトレースベースシミュレーションはインオーダ実行を仮定すれば動作は一意に定まるため，メモリアクセストレースはそれを収得する環境に依存して変化しない．マルチコアプロセッサにおいてプログラムの動作はキャッシュメモリに依存して変化する．そのため，マルチコアプロセッサアーキテクチャに対するトレースベースシミュレーションは，メモリアクセストレースを収得する環境とシミュレータ上で再現する環境が異なると原理的に正しい動作をしていない．本章では，メモリアクセストレース収得時の想定するキャッシュアーキテクチャが違うとき，それらのアクセストレースを入力とするキャッシュシミュレータの出力がどう異なるか比較する．

5.1 比較実験

サイクルアキュレートなシミュレータMARSS[10]を使用し，Splash-2ベンチマーク[17]のアプリケーションの

アクセストレースを収得した．使用したアプリケーションは FFT, LU, CHOLESKY, RADIX である．アクセストレースは，キャッシュメモリとして構成 $(32, 32, 1)^2$ を想定したときと構成 $(1024, 1024, 32)^2$ を想定したときの 2 種類のアクセストレースを得た．これらのアクセストレースをトレースベースキャッシュシミュレータに入力し，動作時間を測定し，出力として図 5(a),(b),(c),(d),(e) の 5 つの状況が発生した回数を数えた．キャッシュシミュレータを C 言語で実装した．使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 32, ブロックサイズが 32Byte, 連想度が 1 の構成 $(32, 32, 1)^2$ とセット数が 1024, ブロックサイズが 1024Byte, 連想度が 32 の構成 $(1024, 1024, 32)^2$ である．アクセストレースを入力として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．2 シミュレータの出力である図 5(a),(b),(c),(d),(e) の 5 つの状況が発生した回数の一部を表 2 に示す．

メモリアクセストレースを入手するときに想定したキャッシュ構成 A と，メモリアクセストレースを入力してトレースベースシミュレーションで想定するキャッシュ構成 B が一致するとき，その出力結果はサイクルアキュレートシミュレーションと同等であると考えられる．表 2 の太字はサイクルアキュレートシミュレーションで得られる結果と同等の精度の値である．大きなキャッシュメモリを想定してメモリアクセストレースを入手したとき，小さなキャッシュメモリをシミュレータ上で再現してシミュレートすると出力結果は最大で全命令数の 2.98%の誤差を持つ．同様に，小さなキャッシュメモリを想定してメモリアクセストレースを入手したとき，大きなキャッシュメモリをシミュレータ上で再現してシミュレートすると出力結果は最大で全命令数の 2.79%の誤差を持つ．メモリアクセストレースを入手するときには，小さなキャッシュメモリを想定したほうがトレースベースシミュレーションの誤差が少なくなると考えられる．

6. おわりに

本稿ではメモリアクセストレースを入手するときに想定したキャッシュ構成とトレースベースシミュレーションで想定したキャッシュ構成が異なるときのトレースベースシミュレーションの精度を評価した．トレースベースシミュレーションのためのメモリアクセストレースの収得環境としては，小さいキャッシュメモリを想定したほうがよりサイクル精度シミュレーションに近い結果がでるといえる．

謝辞

本研究の一部は，早稲田大学特定課題研究 (課題番号 2012A-603), NEC からの研究費，科研費 (特別研究員奨励費) 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] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, “Evaluation techniques for storage hierarchies,” *IBM System Journal,* vol. 9, no. 2, pp.78–117, 1970.

[9] 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.

[10] A. Patel, F. Afram, S. Chen, and K. Ghose, “MARSS: A full system simulator for x86 CPUs,” in *Proc. DAC 2011,* 2011, pp. 1050–1055.

[11] 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.

[12] P. Stenstrom, “A survey of cache coherence schemes for multiprocessors,” *Computer,* vol. 23, no. 6, pp. 12–24, 1990.

[13] R. A. Sugumar “Set-associative cache simulation using generalized binomial trees,” *ACM Trans. Computer Systems*, vol. 13, no. 1, pp. 32–56, 1995.

[14] 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.

[15] 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.

[16] M. Vega, J. Sanchez, R. Montafia, and F. Zarallo, “Simulation of cache memory systems on symmetric multiprocessors with educational purposes,” in *Proc. the I International Congress in Quality and in Technical Education Inovation,* 2000, pp. 47–59.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proc. ISCA 95,* 1995, pp. 24–36.

Vol.2013-SLDM-160 No.15
Vol.2013-EMB-28 No.15
2013/3/13

表 2 2 種類のアクセストレースを入力としたときそれぞれの図 5(a),(b),(c),(d),(e) の 5 つの状況が発生した回数.

		シミュレートする構成	構成 $(32,32,1)^2$ のアクセストレースを入力					構成 $(1024,1024,32)^2$ のアクセストレースを入力				
			(a)	(b)	(c)	(d)	(e)	(a)	(b)	(c)	(d)	(e)
Splash-2	FFT	$(32,32,1)^2$	**681432**	**7268**	**209197**	**431920**	**68565**	683491	6799	210233	434257	71612
		$(1024,1024,32)^2$	896559	182	1156	499717	768	**898290**	**1052**	**1181**	**504164**	**1705**
	LU	$(32,32,1)^2$	**1602557**	**6953**	**1051364**	**1275570**	**96778**	1589621	8674	1040201	1266630	94228
		$(1024,1024,32)^2$	2657565	2089	1220	1369259	3089	**2637172**	**192**	**1132**	**1360017**	**841**
	CHOLESKY	$(32,32,1)^2$	**8132744**	**18162**	**2365673**	**3899447**	**613548**	8427372	23954	2487083	3883425	623596
		$(1024,1024,32)^2$	10511987	3282	1310	4506641	6354	**10935066**	**2078**	**1265**	**4502196**	**4825**
	RADIX	$(32,32,1)^2$	**18785609**	**7752**	**2297231**	**4079264**	**929331**	18827840	7222	2327800	4080864	947726
		$(1024,1024,32)^2$	21087451	1940	1201	5003735	4860	**21159835**	**1791**	**1236**	**5023937**	**4653**