情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-ARC-196 No.24
2011/7/28

データ値の局所性を利用したライン共有キャッシュの提案

岡　慶太郎[†1]　福本尚人[†1]
井上弘士[†2]　村上和彰[†2]

キャッシュメモリの容量を有効に利用することで，キャッシュミス率を大幅に削減する手法としてライン共有キャッシュを提案する．従来型キャッシュでは，アドレスの異なる 2 つのデータが同一の値を有する場合においても，これらは異なる記憶領域に保存される．これに対し，ライン共有キャッシュは，キャッシュ内に同一の値を有するデータが多く存在することに着目し，これらのデータを 1 か所の記憶領域に保存することで，キャッシュメモリを有効利用する．ベンチマークプログラムを用いた定量的評価の結果，L2 キャッシュ容量 1MB を有するライン共有キャッシュは従来型キャッシュの L2 キャッシュミス率を最大 18 ポイント削減できることを確認した．

Line Sharing Cache Based A Frequent Value Locality

Keitarou Oka,[†1] Naoto Fukumoto,[†1] Koji Inoue[†2]
and Kazuaki Murakami[†2]

This paper proposes line sharing cache(LSC) which reduce miss rate of last level cache without increasing the size of cache memory. The conventional cache memory stores blocks in lines based on adress.In conventional cache memory,if two blocks is the same value and the different adress,these data will be stored in diffrecnt cache blocks.We observed the large amount of block in conventional cache filled with frequnet value.Therefore the conventinal cache don't utlize area of cache memory effectively.The other hand,line sharing cache stores blocks which have the same value in the same block. LSC allows greater amounts of data to be stored leading to substantial reduction in miss rate(0-18 point).

†1 九州大学大学院システム情報科学府
Graduate School of Infomation Science and Electrical Engineering
†2 九州大学大学院システム情報科学研究院
Faculty of Infomation Science and Electrical Engineering

1. はじめに

現在，1 チップに複数のコアを搭載するマルチコア・プロセッサが主流となっている．チップ内スレッドレベル並列処理により高い演算性能を達成できるためである．しかしながら，パッケージ制約による I/O ピンの制限等によりオフチップ・メモリバンド幅はコア数に対してスケールしない．また，主記憶として使用される DRAM の動作速度はプロセッサ速度と比較して極めて遅い．その結果，プロセッサとオフチップメモリの性能差拡大（いわゆる，メモリウォール問題）がより深刻化する．この問題を解決するために，多くのマイクロプロセッサでは大容量なラストレベル・キャッシュ(LLC) を搭載している．例えば，Intel の Core i7 では 8MB，AMD の Barcelona においては 2MB の LLC を搭載しており，オフチップ・アクセス回数の削減を目的に年々大容量化する傾向にある．

一般に，キャッシュメモリの大容量化はチップ面積の増加を伴うため，その結果として製造歩留まりが低下する．また，CMOS LSI のリーク電力はトランジスタ数に比例するため，プロセッサの消費電力が増加すると言った問題を引き起こす．キャッシュ面積を一定に保ちつつ，その性能を改善する代表的なアプローチとしてデータ・プリフェッチが挙げられる．プロセッサが将来参照するであろうデータを事前にキャッシュメモリへロードすることにより，メモリアクセス・レイテンシを隠蔽できる．しかしながら，プリフェッチ精度が低い場合にはメモリバンド幅を浪費するため，性能低下や消費エネルギーの増大を招く恐れがある．したがって，例えばメニーコアのように今後のコア数増加を見据えた場合，限られたキャッシュ容量を効率良く活用し，面積増大やバンド幅の浪費を招くことなくオフチップ・アクセスを削減可能な抜本的対策が必要となる．

そこで本論文では，大幅な面積増加を伴うことなくキャッシュミス率を削減する新しいアーキテクチャとして**ライン共有キャッシュ（LSC：Line Sharing Cache)** を提案する．また，ベンチマークプログラムを用いた定量的評価を行い，その有効性を明らかにする．LSC では，データ値の局所性に着目し，キャッシュメモリの容量を有効利用する．ここで，データ値の局所性とは，メモリアドレスが異なる複数のデータが全く同一の値を有すると言う性質である．データ値の局所性が高い場合，キャッシュメモリ中に記憶された複数のブロックが同一の値を有する確率が高いことを意味する．従来のキャッシュメモリでは，参照アドレスによりブロックの格納場所 (キャッシュ・セット) が決定され，1 つのラインに 1 つのブロックのみを格納することができる．本稿では，キャッシュメモリのレベル間で取り交わすデータの最小単位を**ブロック**，キャッシュメモリにおけるブロック格納場所を**ライン**と呼ぶ．

Vol.2011-ARC-196 No.24
2011/7/28

これに対し，提案する LSC では，同一の値を有する複数ブロックが検出された場合には，これらが 1 つのラインを共有することを許す．つまり，複数ブロックが単一ラインに格納されることになり，従来手法と比較してキャッシュ容量を効果的に活用することができる．ベンチマーク・プログラムを用いた定量的評価の結果，データアレイの容量が 1MB の LSC キャッシュは，最善の場合で 8MB の従来型キャッシュメモリと同等のヒット率を達成できることを確認した．

本稿の構成を以下に示す．第 2 節でデータ値の局所性について説明し，第 3 節で提案方式であるライン共有キャッシュの詳細を述べる．第 4 節ではライン共有キャッシュの性能評価を行い，その有効性を明らかにする．最後に第 5 節で本稿をまとめる．

2. データ値の局所性

2.1 同一データ値の書込み頻度分析

本節では，プログラム実行におけるデータ値の局所性を分析する．データ値の局所性が高い場合には，メモリに対して同一データの書込みが頻発すると予想される．そこで，ベンチマーク・プログラムの実行における書込みデータ値の偏りを調査する．ここで，書込みデータ値とは，プロセッサがストアする 64bit データの値である．実験環境の詳細は第 5.1 節で説明する．各プログラムの実行において書込みデータ値毎に書込み回数を測定し，降順に累計した結果を図 1 に示す．各図において，横軸は書込みデータ値の種類数を総書込みデータ値の種類数で正規化した値，縦軸は累計書込み回数を総書込み回数で正規化した値である．これらの結果より，書込みデータ値の種類数の高々20%が総書込み回数に占める割合は，*Cholesky* で約 75%，*Barnes* では約 70%，*FFT* では約 65%，*FMM* では約 50% と極めて高いことが分かる．すなわち，多くのプログラムにおいて，同一の値を頻繁に書込む傾向にあると言える．

2.2 キャッシュメモリにおけるデータ値の局所性分析

キャッシュメモリは書込み要求を受付けるとセット中のいずれかのラインにブロックを書込む．データ値の局所性が高いプログラムを実行した場合，同一値を有するブロックがキャッシュ内に多数存在することになる．そこで，プログラム実行におけるキャッシュメモリ内のデータ値の局所性を調査した．まず，第 5.1 節の表 1 で示すプロセッサ構成にて，L2 キャッシュメモリへのアクセストレースを取得する．ここで，L2 キャッシュのブロックサイズは 64B，容量は 1MB とする．次に，採取したメモリアクセストレースより，式 (1) の値をブロックの置換え毎に取得し平均を取る．この値が低いほどキャッシュメモリ中に同一値を有

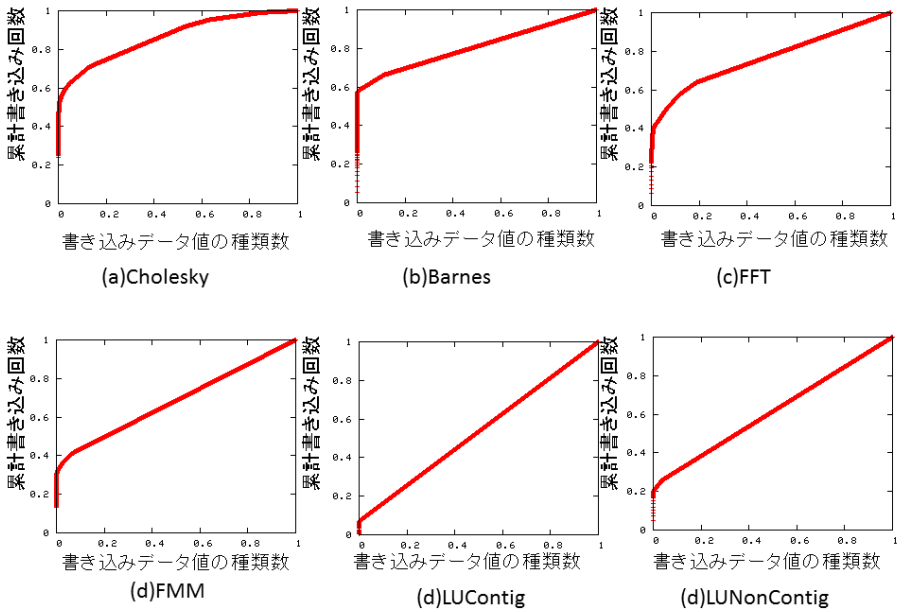

図 1 データ値の局所性

するブロックが多く存在することになる．

$$\frac{\text{ブロック値の種類数}}{\text{キャッシュメモリ中の有効ブロック数}} \tag{1}$$

図 2 に実験結果を示す．キャッシュメモリ内のブロック数に対するブロック値の種類数の割合は，最小で 22%(*Cholesky*)，平均で 46%である．つまり，*Cholesky* の場合，最低限保持すべきブロック値はキャッシュに格納された全有効ブロックの 22%程度であり，残り 78%のブロックはこの 22%のブロックと同一のデータ値を有する．つまり，従来型キャッシュでは同一ブロック値を冗長に記憶しており，同一値を有するブロックを 1 つのラインにまとめて格納することができれば，記憶する情報量は同一のままキャッシュメモリのデータアレイの容量を 5 分の 1 程度に削減できる可能性があることを意味する．

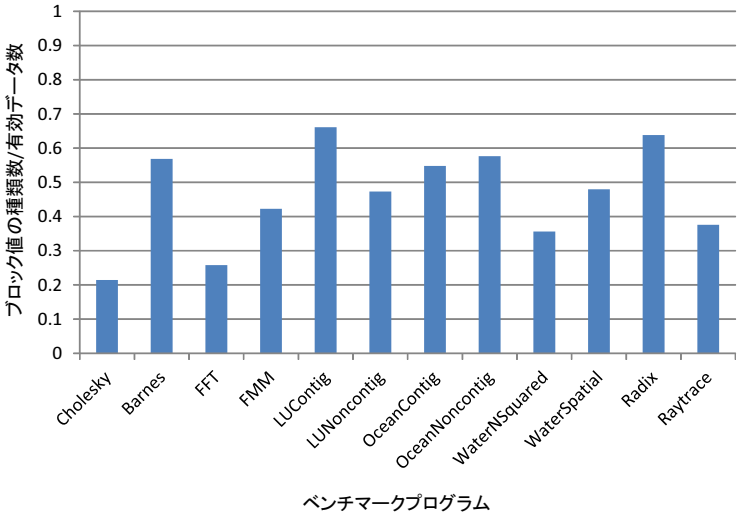

図 2 キャッシュメモリ内のデータ値の局所性

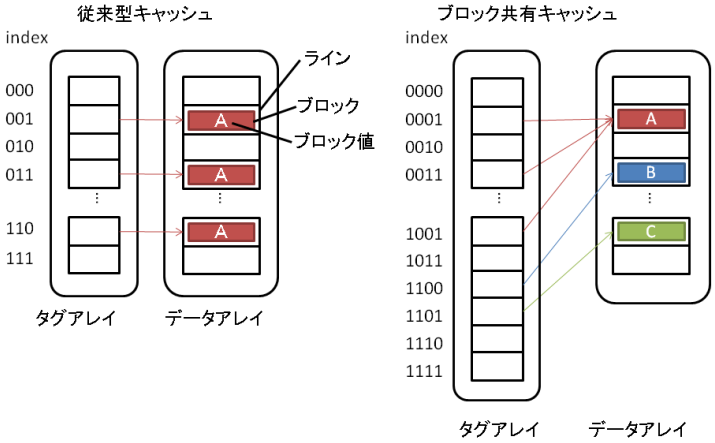

図 3 従来手法と提案手法の比較

3. ライン共有キャッシュ

3.1 基本アイデア

前節で述べたように，データ値の局所性が高い場合，従来型キャッシュでは多数のラインに同一の値を有するブロックが格納される．そのため，キャッシュメモリの容量を有効に利用できないといった問題が生じる．そこで，同一の値を有するブロックの格納場所を 1 つのラインに限定することにより，キャッシュメモリの容量を効率よく利用する手法としてライン共有キャッシュ(Line Sharing Cache, LSC) を提案する．

従来型キャッシュと LSC の概念を図 3 に示す．図中の矢印はタグとラインの対応を示す．従来型キャッシュではタグとラインが一対一対応となる．ここで，インデックスが異なる書込みアクセスが 3 回発生し，これら全てのアクセスにおいて書込みデータは同一の値 A を有する場合を考える．この例では，書込みデータを記憶するために従来型キャッシュでは 3 つのラインが必要になる．一方，LSC では同一の値を有するブロックは 1 つのラインを共有する．

図 3 で示すように，複数のタグが一つのラインに対応しており，値 A を有する 3 つのブロックを格納するラインは 1 つとなる．このように，LSC は複数のブロックがラインを共有することで，従来方式と比べてより多くのブロックをデータアレイに格納できる．その結果，キャッシュメモリの容量不足によるキャッシュミスを削減することが可能となる．

3.2 マイクロアーキテクチャ

図 4 に連想度 2 の LSC の全体像を示す．LSC はタグ・ポインタアレイとデータアレイからなる．タグ・ポインタアレイの各エントリは，有効ビット，タグ，ならびに，ポインタから構成される．有効ビットとタグは従来のキャッシュメモリと同様である．ポインタに関しては，データアレイに格納されたブロックを指定するための**行番号**と**列番号**から成る．一方、データアレイは複数のラインで構成され，各ラインにはブロックが格納される．ここで，以下の用語を定義する．

- ポインタセット：タグ・ポインタアレイにおいて 1 つのインデックスに対応する複数のタグ・ポインタから成るグループ

Vol.2011-ARC-196 No.24
2011/7/28

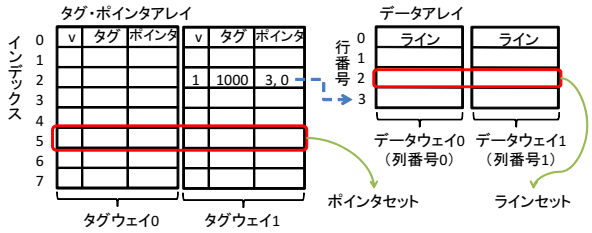

図 4 LSC のアーキテクチャ

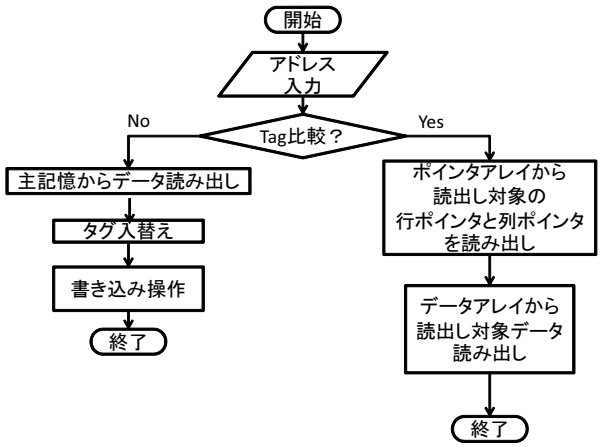

図 5 読出しの実行フロー

- ラインセット：データアレイにおいて同一行番号を有するラインの集まり
- タグ連想度：1 ポインタセットあたりのポインタの数（従来型キャッシュの連想度に相当）
- ライン連想度：1 ラインセットあたりのライン数（LSC 特有の連想度でありタグ連想度と同一である必要はない）

ここで，図 4 は，タグ連想度 2，ライン連想度 2 において，ラインセット数がポインタセット数の半分の場合である．

3.3 動　　作

LSC の動作について説明する．図 5 に read 要求に対する LSC の動作フローを示す．まず，図 6 のように参照アドレスのインデックスを用いてタグ・ポインタアレイのポインタセットからタグを読出す．そして，参照アドレスのタグフィールドと読出したタグが一致するか比較する．一致した場合（キャッシュヒット），行ポインタと列ポインタが指すデータアレイから該当ブロックを読出しアクセスを終了する．タグが一致しない場合（キャッシュ・ミス）には，下位メモリ階層からデータを取得し，ブロック置換のためのタグ情報の更新を行った後に，上層メモリ階層にデータを送信すると同時に書込み操作を行う．これら上位階層へのデータ送信と書込み操作は並列に処理できるため，書込み操作に要する時間はプロセッサから隠蔽可能である．

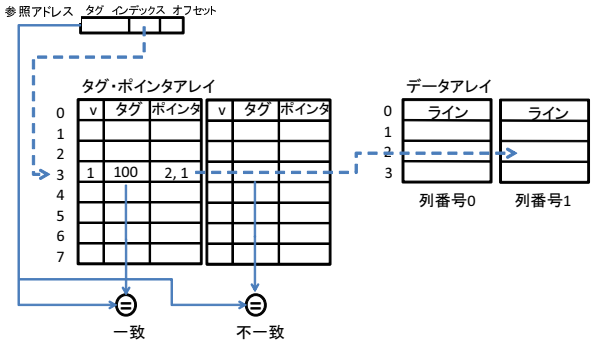

図 6 LSC の読み出し動作（タグ一致の場合）

書込み操作の動作フローを図 7 に示す．図 8 のように書込みブロックの行ポインタと一致するデータアレイからラインセットを読出し，書込みブロック値とラインセットのブロック値が一致するか否かを判定する．一致する場合，タグ・ポインタアレイのポインタフィールドの更新を行う．一致しない場合，ラインセット内で書込み追出しブロックを決定する．追出しデータを主記憶に書戻し，追出し対象ラインに書込みデータを書込む．そして，追出しラインの行・列番号を有するタグをすべて無効化し，書込みデータに対応する行・列ポインタを更新する．

次に，図 7 を用いて write 要求に対する LSC の動作フローについて説明する．まず，Read 要求の動作時と同様に，参照アドレスを元にタグの一致・不一致を判定する．タグが一致する場合は，データアレイから読み出したデータと書込みデータを統合したデータを元に書込み操作を行う．タグが一致しない場合は，タグの入換えを行い，主記憶から読み出したデー

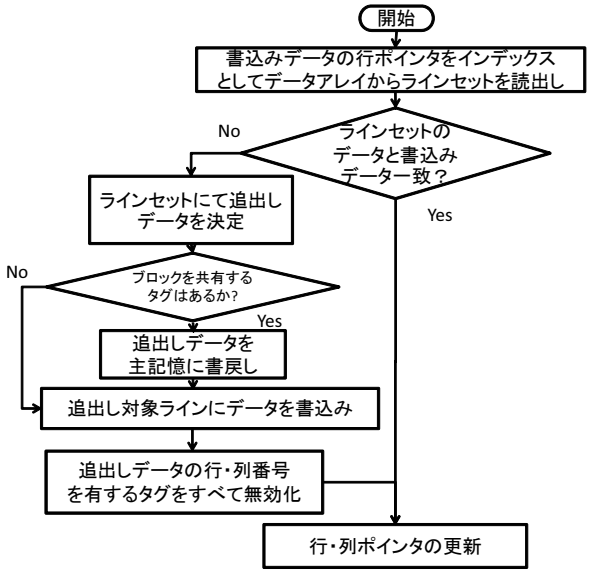

図 7 書込み操作の実行フロー

タと書込みデータを統合し，図 7 のフローで書き込み操作を行う．ここの書込み操作の内，主記憶への書戻しはプロセッサからは隠蔽される可能性がある．

3.4 性能モデリング

LSC の性能モデルを構築には以下の事柄を仮定する．第一に，タグ・ポインタアレイのポインタ部分のアクセス時間を無視する．これはポインタ部分の容量がデータアレイやタグアレイに比べて極めて小さいためである．第二に，ブロック追い出し時に追い出しブロックを共有するタグを検索する時間を考慮しない．これは LSC がタグ検索時間を隠ぺいする事が可能であるためである．

提案方式の性能モデルを式 2 のように定義する．

$$AMAT_{LSC} = HT_{L1} + MR_{L1} \times (HT_{L2\_LSC} + MR_{L2\_LSC} \times MP_{LSC}) \quad (2)$$

- HT_{L1}：L1 キャッシュのアクセス時間 [クロック・サイクル]
- MR_{L1}：L1 キャッシュのミス率
- HT_{L2_LSC}：L2 キャッシュのアクセス時間 [クロック・サイクル]

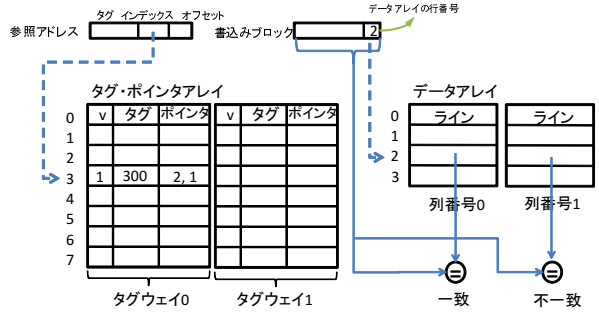

図 8 LSC の書込み操作（書込みブロック一致時）

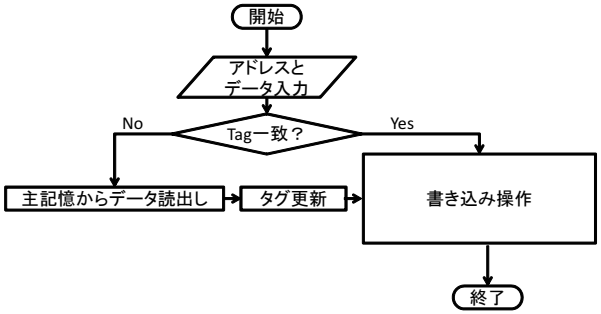

図 9 書込み時の実行フロー

- MR_{L2_LSC}：L2 キャッシュのミス率
- MP_{LSC}：L2 ミスペナルティ [クロック・サイクル]

プロセッサから読出しまたは書込み要求が発生した際，まず L1 キャッシュメモリにアクセスするため HT_{L1} を要する．L1 キャッシュミスした場合，L2 キャッシュメモリにアクセスしさらに HT_{L2_LSC} 費やす．ここで，HT_{L2_LSC} は式 3 で示すように読出しと書込みで区別する．

$$HT_{L2\_LSC} = P_{read} \times AT_{read} + P_{write} \times AT_{LSC\_write} \quad (3)$$

- P_{read}：読出し確率（L2 読出し回数/L2 アクセス回数）
- P_{write}：書込み確率 (L2 書き込み回数/L2 アクセス回数)

- AT_{read}：L2 キャッシュの読出し時間
- AT_{LSC_write}：LSC における L2 キャッシュの書込み時間

L2 キャッシュメモリに対する書込みでは従来方式とは異なり AT_{LSC_write} 要する．AT_{LSC_write} を式 4 に示す．そこで，データアレイから書込みブロック値を検索する．データアレイを全検索をする場合，検索に多大な時間とエネルギーがかかる．したがって，検索するブロックの個数を減らす必要である．

$$AT_{LSC\_write} = ST_{data} + DNER \times AT_{write} \quad (4)$$

- ST_{data}：ラインセット中のデータを検索する時間 [クロック・サイクル]
- $DNER$：データ非存在率
- AT_{write}：書込みアクセス時間 [クロック・サイクル]

LSC は L2 キャッシュメモリにブロックを書き込む場合，ラインセット中から書込みブロック値を検索するために ST_{data} 費やす．ブロック値を発見できない場合はブロック値をブロックに書込むためにさらに，AT_{write} 要する．データ非存在率 (DNER) はブロック値を発見できない割合であり，次のように定義する．

$$DNER = \frac{\text{書込みブロックをデータアレイから発見できない回数}}{\text{L2 キャッシュに対する合計書込み回数}} \quad (5)$$

L2 キャッシュミスした場合，主記憶にアクセスし MP を費やす．MP は，式 (5) のように書込み時には主記憶アクセス時間のみであり，読出し時にはキャッシュ・フィルのために主記憶アクセス時間と AT_{LSC_write} の和となる．

$$MP = MMAT + P_{read} \times AT_{LSC\_write} \quad (6)$$

3.5 LSC の利点と欠点

LSC は複数のブロックがラインを共有することで，従来方式に比べより多くのブロックをデータアレイに格納する．その結果，L2 キャッシュミス率を削減することが可能になる．さらにキャッシュメモリ中に書込みブロック値が存在する場合，データアレイに書込みを行わないため，データアレイに対する書込み回数を削減できる．

しかしながら LSC には 3 つの問題が存在する．読出し時にはまず，タグ・ポインタアレイでデータの場所を読出した後に，データアレイの該当データにアクセスする．その結果，タグ・ポインタアレイとデータアレイにアクセスするので，従来のキャッシュメモリに比べ読出しに時間が長くかかる．第 2 に，ラインセット内のブロックを追い出すときに問題を生じる．データブロックを追い出すときには，追い出しブロックに対応するポインタをポインタアレイから検索する．タグ・ポインタアレイのポインタセット数は非常に多いので，ポインタの検索には多くの時間とエネルギーを要する．またブロックを追い出すとブロックを共有しているタグをすべて無効化するためキャッシュミス率を増加させる要因となる．第 3 に，LSC は下位数ビットが等しいブロックを同じラインセットに格納する．ブロック値のマッピングが一様にならない場合，特定のラインセットに多くブロックが割り当てられることになり，ブロックの追い出しが頻発する．

4. 評　　価

4.1 評 価 方 法

表 1 プロセッサの構成

コアの構成	8 コア，イン・オーダ
L1 命令キャッシュ	32KB，2-way，64B lines, 1 clock cycle MSHR 8
L1 データキャッシュ	32KB,2-way,64B llines 1 clock cycle MSHR 32
L2 キャッシュ	1M,2M,4M,8M,32M, 8-way,64B,32B,16B,8B lines 12 clock cycles MSHR 92
L1-L2 間共有バス幅	64B
L2-主記憶間共有バス幅	16B
主記憶レイテンシ	300 clock cycles
Miss Status Buffer	エントリ数:20,1 clock cycle

表 2 キャッシュメモリシミュレータの設定 (90nm)

ブロックサイズ	64B	32B	16B	8B
バス幅	256B	128B	64B	32B

キャッシュのミス率の導出法

ライン共有キャッシュの L2 キャッシュミス率 MR_{L2_LSC} を従来型キャッシュの容量とミス率の関係から求める．LSC は従来型キャッシュに比べ容量を有効利用するため，ミス率の評価には実際の容量を従来型キャッシュの容量に変換した見かけ容量を用いる．この見かけ容量におけるミス率を LSC のミス率とする．

ここで，実際の LSC の容量を「LSC 容量」，LSC 容量を従来型キャッシュに近似した場合の容量を「見かけ容量」と定義すると，見かけ容量は LSC 容量と平均圧縮率を用いて式 7 のように表せる．

$$C_{bsc\_effective} = \frac{C_{LSC}}{CR} \tag{7}$$

- $C_{bsc\_effective}$:見かけ容量
- C_{lsc}:LSC 容量

平均圧縮率 (Compression Rate,CR) を式 8 のように定義する．平均圧縮率 CR はブロックの追出しごとに cr_i を求め，平均した値である．

$$CR = \frac{1}{n} \times \sum_{i=1}^{n} cr_i \tag{8}$$

$$cr_i = \frac{\text{ブロックの値の種類}}{\text{有効ブロック数}} \tag{9}$$

n : 全 replace 回数

つぎに，見かけ容量により，MR_{L2_LSC} を求める．見かけ容量から MR_{L2_LSC} を求めるために関数 f, 関数 g, 関数 h を定義する．関数 f は 10 のように，見かけ容量と MR_{L2_LSC} の関係を表す．

$$MR_{L2} = f(C_{LSC\_effect}) \tag{10}$$

見かけ容量は 7 から CR と LSC 容量の関数であるため，11 のように，g を定義する．

$$C_{LSC_effect} = \frac{C_{LSC}}{CR} = g(C_{LSC}, CR) \tag{11}$$

CR は LSC 容量の関数であり，この関数を h とすると 12 のように表せる．

$$CR = h(C_{LSC}) \tag{12}$$

関数 f,g,h を用いると LSC 容量から L2 ミス率を求める事ができる．

$$MR_{L2} = f \circ g \circ h(C_{LSC}) \tag{13}$$

MR_{L2} をもとめる手順を以下に示す．まずマルチコアシミュレータ M5[1] により，L2 キャッシュのトレースを取得する．このときの評価環境を表 1 に示す．対象ベンチマークは splash2[2] の中から選択した．

つぎに，L2 キャッシュのアクセストレースから CR を求める．そして，LSC 容量と CR の近似曲線，LSC 容量と L2 ミス率の近似曲線を求め，それぞれ関数 h，関数 f とする．その後，式 13 を用いて，MR_{L2_LSC} を求める．

従来型キャッシュメモリにおけるキャッシュミス率も，もマルチコアシミュレータ M5 により求めた．

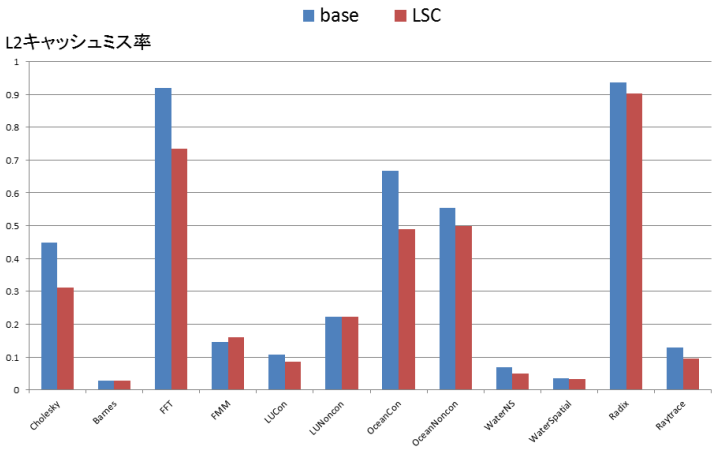

図 10 データアレイのエントリ数が等しい場合の L2 キャッシュミス率

L1 レイテンシ，L2 レイテンシ

L1 キャッシュメモリと SRAM L2 キャッシュメモリのアクセスレイテンシはキャッシュメモリシミュレータ cacti[3] より求めた．cacti の設定を 2 に示す．

L1 ミスペナルティ求め方

式 (2) の L1 ミス率を乗算している項に値を代入することで，L1 ミスペナルティを求めた．ただし，ポインタアレイの容量が小さいため，ポインタアレイのアクセス時間を無視した．

4.2 評価結果

図 10 に L2 キャッシュミス率を示す．横軸はベンチマークプログラム名，縦軸は L2 キャッシュミス率である．2 つのバーは左から従来キャッシュメモリ，LSC を示す．ここで，従来キャッシュメモリと LSC のデータアレイのエントリ数は等しい．多くのベンチマークプログラムで，キャッシュミス率を削減できている．FFT や OceanCon ではそれぞれ 19 ポイント，18 ポイント程度のミス率削減を達成している．この結果より LSC はデータアレイ容量を増やすことなくミス率を削減できることが分かった．

図 11 ミス率一定時の面積削減効果（従来キャッシュメモリのサイズ 8MB）

図 11 に 8MB の従来キャッシュにおけるミス率を達成するために必要な記憶容量の値を示す．横軸はベンチマークプログラム名，縦軸はキャッシュメモリを構成する記憶容量のサイズを示す．バーの内訳は，データアレイの容量とタグの容量である．ただし，LSC のタグ容量にポインタの容量は含まれない．すべてのベンチマークプログラムにおいて，データアレイの容量を削減することにより，大幅に記憶容量を削減できている．特に削減量の多い Cholesky では，約 67%の記憶容量削減を達成した．これらの結果より，従来キャッシュと比較して，LSC は面積削減効果が高いことが分かる．

図 12 に，従来キャッシュメモリで正規化した LSC の L1 ミスペナルティを示す．この L1 ミスペナルティは，式 (hoge) から L1 ヒット時間を差し引いて求めた．すべてのベンチマークプログラムで，L1 ミスペナルティを削減できている．特に Cholesky, OceanContig, OceanNoncontig では L1 ミスペナルティを 20%程度削減できている．

5. ま と め

L2 キャッシュメモリの容量を有効に活用する手法としてデータ値の局所性に着目し，ブロック共有キャッシュを提案した．その後，BSC の性能モデルを用いて性能評価を行った．BSC の性能を決める要因は「容量とミス率の関係」と「データ非存在率」である．また，ブロックサイズを減少する事で L2 キャッシュミス率を大幅に削減できる．

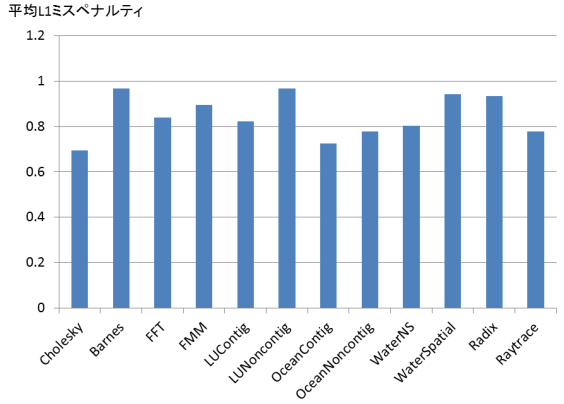

図 12 Read 時の正規化 L1 ミスペナルティ

6. 今後の予定

今回の評価ではブロック共有キャッシュのミス率とデータ非存在率を従来手法の容量を増加する事で近似した．しかしながら，この近似には 2 つの問題点が存在する．第一に近似手法では提案手法のブロック置き換えの頻度を正確に求められていない．ブロック共有キャッシュでは下位数ビットが等しいブロックをブロックセットに格納する．データのマッピングが一様にならない場合，特定のブロックに多くのデータが割り当てられることになるため，ブロックの置き換えが頻発する．第二にブロック置き換え時にタグを無効化において，共有するタグの影響を無視している．BSC ではブロック置き換え時に追い出しブロックを共有する全てのタグを無効化するのに対し，近似手法では追い出しブロックに対応する一つのタグのみを無効化する．そのため正確なミス率が得られていない．そこで，今後はデータを一様にマッピングする手法を提案した後に，ブロック共有キャッシュをシミュレータ上に実装し定量的評価を行う．

謝辞 日頃から御討論頂いております九州大学安浦・村上・松永・井上・アシル・杉原研

Vol.2011-ARC-196 No.24
2011/7/28

究室ならびにシステム LSI 研究センターの諸氏に感謝します．本研究は主に九州大学情報基盤研究開発センターの研究用計算機システムを利用しました．なお，本研究は，一部、独立行政法人新エネルギー・産業技術総合開発機構 (NEDO)「極低電力回路・システム技術開発（グリーン IT プロジェクト）」「低消費電力メニーコア用アーキテクチャとコンパイラ技術」の支援による．

参 考 文 献

1) Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A. and Reinhardt, S.: The M5 simulator: Modeling networked systems, *Micro, IEEE*, Vol.26, No.4, pp.52–60 (2006).
2) Woo, S., Ohara, M., Torrie, E., Singh, J. and Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations, *Proceedings of the 22nd annual international symposium on Computer architecture*, ACM, pp.24–36 (1995).
3) Thoziyoor, S., Muralimanohar, N., Ahn, J. and Jouppi, N.: CACTI 5.1, *HP Laboratories, April*, Vol.2 (2008).