情報処理学会論文誌　コンピューティングシステム　Vol.5　No.4　36–47 (Aug. 2012)

# データ値の局所性を利用したライン共有キャッシュ

岡 慶太郎[1,a)]　佐々木 広[2]　阿部 祐希[1]　井上 弘士[2]　村上 和彰[2]

受付日 2012年1月17日, 採録日 2012年5月15日

**概要**：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 カ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，理想的なライン共有キャッシュは従来型キャッシュに対し大幅な性能向上を達成することを確認した．

**キーワード**：キャッシュメモリ，データ値の局所性

# Line Sharing Cache Based on Frequent Value Locality

Keitarou Oka[1,a)]　Hiroshi Sasaki[2]　Yuki Abe[1]
Koji Inoue[2]　Kazuaki Murakami[2]

Received: January 17, 2012, Accepted: May 15, 2012

***Abstract:*** This paper proposes a new LLC architecture called line sharing cache (LSC) which reduces the number of misses without increasing the size of the cache memory. The key observation used in LSC is that a large amount of lines in conventional caches are filled with frequent values. Conventional caches store a cache line to a set of lines based on their addresses, and therefore many lines which have the exact same value are stored all over the cache memory. While this architecture is well designed to effectively manage the data, it wastes many transistors by having redundant data values. LSC stores lines which have the identical value in a single line and allows greater amounts of data to be stored, leading to substantial reduction in the number of cache misses. The evaluation results show the potential performance improvement of the LSC over the conventional L2 cache architecture.

***Keywords:*** cache memory, value locality

## 1. はじめに

現在，1 個の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．チップ内での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，I/O ピン・ボトルネックなどによりオフチップ・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きわめて低い．そのため，プロセッサとオフチップメモリの性能差拡大がより深刻化するといった問題（いわゆる，メモリウォール問題）が生じる．これを解決するために，近年多くのマルチコア・プロセッサは大容量のラスト・レベル・キャッシュ（LLC: Last Level Cache）を搭載しており，その重要性はコア数の増加とともに年々高まっている．

一般に，LLC の実装においてはオフチップ・アクセス回数の削減が最重要課題となる．そのため，高い連想度かつ大容量なキャッシュ構成を採る場合が多い．特に，大容量化は容量性ミスを削減するきわめて直接的かつ効果的な

[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, Kyushu University, Fukuoka 819–0042, Japan
[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, Kyushu University, Fukuoka 819–0042, Japan
[a)] oka@soc.ait.kyushu-u.ac.jp

アプローチであり，高性能プロセッサのLLC容量は増加傾向にある．たとえば，商用プロセッサであるIntel社のCore i7では4 MBから12 MBもの大容量LLCを搭載している[4]．しかしながら，LLCの大容量化は多くのトランジスタ資源を必要とするため，チップ面積の増大やリーク消費電力の増加を招く．したがって，これらの問題をともなうことなく容量性ミスを削減できる新しいキャッシュ構成法の考案が求められる．

通常，従来のキャッシュ構成法では，参照アドレス（またはタグ）とキャッシュラインが1対1に対応する．すなわち，異なるアドレスを有するデータは，その値そのものに関係なく，異なるキャッシュラインに格納される．その結果，キャッシュ内には，**まったく同じデータ値を有する複数のラインが存在する**といった状況が発生する．ベンチマーク・プログラムを用いた定量的解析を行った結果，一意なデータ値を有するラインは，1,000回の書き込みデータに対して，10%以下の場合があることを確認した（すべてのベンチマークの平均では26%）．これは今後1,000回の書き込みが起こるまでに参照されるであろうデータのうち，90%以上のデータは冗長であることを意味する．このように，同一のデータ値を有するラインが多数存在する性質を**データ値の局所性**と定義する．このデータ値の局所性が高い場合，キャッシュ内には同一の値を有するラインが数多く存在しており，キャッシュ容量を無駄に浪費していることを意味する．

そこで本稿では，キャッシュライン単位でのデータ値の局所性を活用した新しいキャッシュ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，提案方式による性能向上の可能性を解析する．従来型キャッシュとは異なり，LSCでは複数のタグが単一のキャッシュ・ラインに対応することを許す．そして，アドレスは異なるが同一のデータ値を有するラインを1カ所の格納場所に割り当てることで，冗長なラインの記憶を回避する．これにより，従来型キャッシュと同じミス率を維持しつつ，実装すべきキャッシュラインのエントリ数を削減できる．これに加え，節約したトランジスタ資源を用いてタグエントリ数を増やす．その結果，同一面積制約下において従来型キャッシュよりも多くのタグエントリを格納することが可能となり，大幅な容量性ミスの削減を実現する．

本稿の構成を以下に示す．2章でキャッシュラインにおけるデータ値の局所性について説明し，ベンチマーク・プログラムを用いた定量的な解析を行う．次に，3章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．4章では性能評価を行い，その有効性を明らかにする．5章で関連研究について説明し，最後に6章で本稿をまとめる．

## 2. ラインにおける値の局所性

本章ではプログラムが有する潜在的なデータ値の局所性を分析する．ベンチマーク・プログラムSPEC CPU2000を実行しL2キャッシュに対するアクセストレースを1,000回の書き込みアクセスごとに集計する．ただし，L2キャッシュはLLCでありL2キャッシュ容量は64 MB，ラインサイズは64 Bである[*1]．次に，1,000回のL2に対する書き込みアクセスごとに一意データ存在率を求め，平均する．一意データ存在率は式(1)で定義されており，1,000回の書き込みデータのうちのユニークなデータが存在する割合を表す．1,000回の書き込みデータに対して一意データ存在率が低い場合，今後参照されるであるデータにおいてデータ値の局所性が高いことを意味する．

$$\frac{\text{1,000回の書き込みデータのうちのユニークな書き込みデータ値の種類数}}{\text{1,000回の書き込みデータ}} \quad (1)$$

**図1**に結果を示す．横軸はベンチマーク・プログラム，縦軸は一意データ存在率を表す．各ベンチマーク・プログラムには4つの凡例が存在しそれぞれデータ長が異なる．つまり，評価はラインサイズを64 Bとしているがラインを複数のデータに分割することで異なるデータの長さに対してデータ値の局所性を分析している．たとえば，4 word（32 B）の結果は前半の32 Bと後半の32 Bを別のラインと考えた場合の結果である．1 wordの結果は評価したベンチマークにおいて一意データ存在比は平均して2.4%ときわめて低い．データの長さが大きくなると一意データ存在率が高くなる傾向にあるが8 wordの結果において9つのアプリケーションのうち，4つのプログラム（mcf，twolf，art，sixtrack）は同一データ存在比率が40%以下となる．これは今後1,000回の書き込みが起こるまでに参照されるであろうデータのうち，半分以上のデータは冗長であることを意味する．8 wordの結果において同一データ存在率の幾何平均は10%である．この結果から同一値を有するラインを1つのキャッシュラインにまとめて格納することで，冗長なデータを格納するトランジスタを利用して実効的なキャッシュ容量を増加させることができる．

## 3. ライン共有キャッシュ

### 3.1 基本アイデア

前章で述べたように，ライン共有キャッシュはLLCに格納された冗長なデータを排除するアーキテクチャである．その実現方法の肝となるアイデアは，タグエントリとデータエントリの対応付けを従来型キャッシュから変更することにある．従来型キャッシュでは各タグは参照アドレスにより決定されるセット内のデータに対応付けられる．しか

[*1] L2キャッシュ以外の構成は4章の表1に示すプロセッサ構成で実行している．

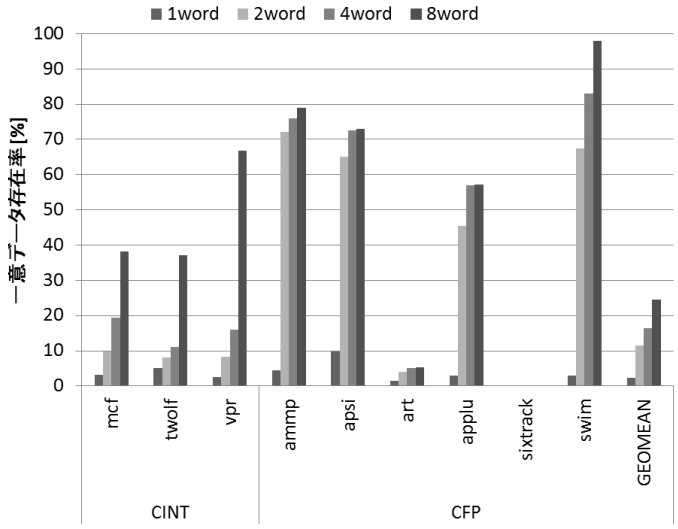


図 1　一意データ存在率
**Fig. 1**　Data uniqueness ratio.

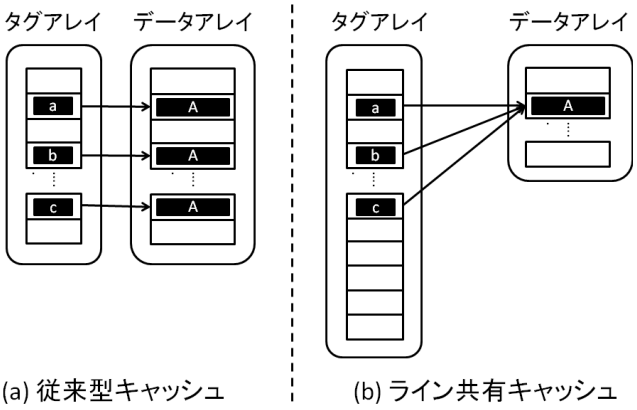


図 2　従来型キャッシュとライン共有キャッシュにおけるタグとデータの対応付け
**Fig. 2**　Tag and data mapping in a conventional cache and Line Sharing Cache.

し，LSC ではタグをデータ値に対応付ける．

従来型キャッシュと LSC のタグエントリとデータエントリの対応付けを**図 2** に示す．タグエントリからデータエントリに対する矢印はタグとデータの対応を表す．図 2 (a) では 3 つのキャッシュライン $a$，$b$，$c$ が同一のデータ値 $A$ をデータアレイのそれぞれ対応するラインに格納している様子を示している．一方，図 2 (b) では LSC はデータ値 $A$ を 1 カ所に格納し 3 つのキャッシュラインの各タグはデータ値 $A$ と対応付けられる．これにより，データアレイの容量を削減し，タグアレイにより多くのトランジスタを使うことで，面積増加をともなうことなく LLC の実効的な容量を増加させることができる．

LSC ではこのようにキャッシュの構造を変更するため，従来型のキャッシュに加えてデータアレイ側において新たな処理を導入する必要がある．具体的には，データアレイでヒット，ミス，および追い出しを考慮することになる．たとえば図 2 (b) で新たに別のキャッシュラインがデータ値 $A$ を格納する場合，このデータ値 $A$ がデータアレイに存在するかを探索する必要がある．これにより，データアレイでのヒットとミスが生じうる．また，データアレイでミスが生じることにより，データ値 $A$ が追い出される可能性がある．この場合，このデータ値と対応付けられている 3 つのキャッシュライン $a$，$b$，$c$ はデータ値を失うことになるため，それぞれ無効化されなければならない．次節以降で，これらの振舞いを実現する LSC の構造およびその動作について詳細に述べる．

### 3.2　構造

**図 3** に LSC のアーキテクチャの構成例を示す．図 2 (b) で示したように LSC はタグアレイとデータアレイから構成される．なお，LSC ではタグとデータが 1 対 1 の関係になく，図 2 からも分かるようにそれぞれのエントリ数は異なっていてもよいため，連想度とセット数はタグとデータでそれぞれ独立に設定できることに注意されたい．図 3 に示す LSC は連想度 2 およびセット数 8 のタグアレイと，連想度 2 およびセット数 4 のデータアレイから構成される場合の例である．また，図中では図 2 (b) のように 1 つのデータが 3 つのラインに共有されている場合を示している．

LSC では以下に示す機構を追加する必要がある．

(1) タグがデータ値に対応付けられているため，タグアレイ側にタグエントリがデータアレイ内のどのラインに対応するかを示す機構が必要となる．

(2) 3.1 節で述べたようにデータ値が追い出された場合にその値に対応付けられていたタグをすべて無効化する必要がある．そこで同一の値を共有しているタグエントリは双方向リストとして管理されている．(a) タグアレイ側のそれぞれのタグエントリに，双方向リスト

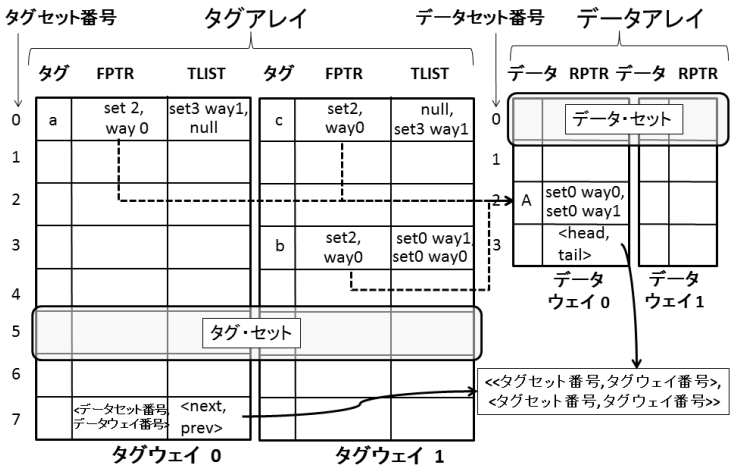


**図 3**　ライン共有キャッシュのアーキテクチャ
**Fig. 3**　Line Sharing Cache architecture.

における次および前のノードを示す機構，および (b) データアレイ側のデータ値には対応する双方向リストの先頭および最後方ノードを示す機構がそれぞれ必要となる．

以下では，タグアレイとデータアレイの構造についてそれぞれ詳細に述べる．

- タグアレイ
  タグアレイの各エントリは有効ビット，ダーティービット，置き換え情報といったキャッシュラインのステータス情報，フォワード・ポインタ（FPTR），タグエントリ・リスト（TLIST）から構成される．FPTR はタグに対応するデータアレイ内のラインを特定するために用いられ，データセット番号（データアレイ内のセット番号）とデータウェイ番号（データアレイ内のウェイ番号）を格納する．図 3 の例ではデータ値 $A$ を共有するタグエントリは，データ値 $A$ を有するラインの場所（データセット番号 2，データウェイ番号 0）を FPTR に格納する．同一のデータ値を持つタグエントリは双方向リストによって管理され，その要素は TLIST である．各 TLIST は自らの前後の要素をタグ・セット番号，データセット番号で特定される格納する．図 3 の例ではデータ値 $A$ を共有するタグは，タグ $a$，$b$，$c$ の順に双方向リストが作られている．
- データアレイ
  データアレイはデータとリバース・ポインタ（RPTR）からなる．RPTR はデータを共有するタグエントリの双方向リストの先頭と末尾を格納する．図 3 の例ではタグ $a$，$b$，$c$ から構成される双方向リストの先頭要素である $a$ の場所（タグセット番号 0，タグウェイ番号 0）および最後方要素である $c$ の場所（タグセット番号 0，タグウェイ番号 1）を格納する．

LSC におけるデータアレイにデータを書き込む際の重要なポイントとして，同一のデータ値を有するラインを探して共有することがあげられる．このため，同一のデータ値を効率良く検索可能にする機構が必要となる．データアレイ内の全ラインと書き込みデータ値を比較することによる検索には多大なレイテンシとエネルギーを要してしまう．そこで，LSC はデータアレイに存在しないデータ値を新たに書き込むときにデータ値を格納できる範囲をデータ値に応じて制限することにより，検索対象となるラインを削減する．説明を簡単にするためにタグアレイの各セットをタグ・セット，データアレイの各セットをデータ・セットと呼ぶ．具体的には，書き込みデータ値の格納可能範囲をデータ値の一部を用いたハッシングにより決定されるデータ・セット内のラインに限定する．ハッシングに用いるデータ値の一部のビット長は $\log 2$（データアレイ内のデータ・セットの数）ビットである．たとえば，データ値 $A$ をデータアレイに書き込む場合を考える．ここでデータ値の下位 2 ビットを用いてハッシングするとデータ値 $A$ はデータセット番号が 2 であるデータ・セットに割り当てられ，当該データ・セットのいずれかのラインに格納される．データ値 $A$ の検索時にはデータ値 $A$ の下位 2 ビットを用いて $A$ が格納されるデータ・セット（この例ではデータセット番号 2 であるデータ・セット）を特定し，当該データ・セット内のエントリに対してのみ比較を行う．

LSC ではタグアレイに FPTR と TLIST，データアレイに RPTR が新たに設けられる．これらを含めたタグアレイならびにデータアレイのビット数は以下のように求められる．

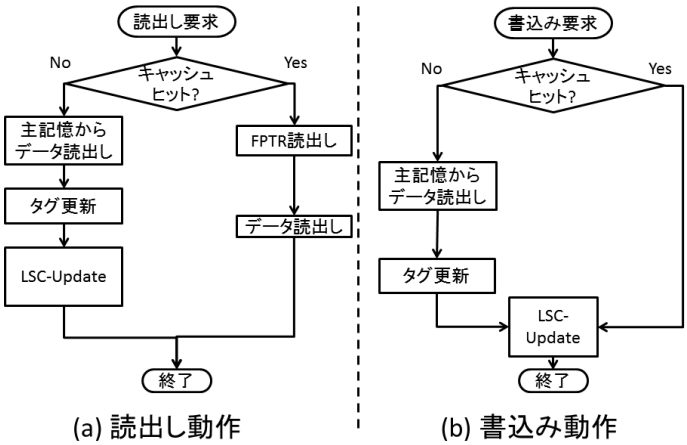


図 4　LSC の読み出しおよび書き込みフロー
**Fig. 4**　Read and write operation flows of LSC.

タグアレイのビット数 = タグエントリのビット数
× タグエントリ数　(2)

データアレイのビット数 = データエントリのビット数
× データエントリ数　(3)

ここで，タグエントリのビット数は次のように求められる．なお，タグ部分のビット数は 64 bit アーキテクチャ，ラインサイズが 64 B の場合の例である．

タグエントリのビット数 = タグ部分のビット数
+ FPTR のビット数
+ TLIST のビット数

タグ部分のビット数 = $64 - \log_2$ タグエントリ数
$- \log_2 64$

FPTR のビット数 = $\log_2$ データエントリ数

TLIST のビット数 = $\log_2$ タグエントリ数 $\times 2$

ただし，TLIST は双方向リスト内，自らの前後の要素を格納するために 2 倍している．データエントリのビット数は次のように求められる．

データエントリのビット数 = データ部分のビット数
+ RPTR のビット数

データ部分のビット数 = 1 ラインのビット数

RPTR のビット数 = $\log_2$ タグエントリ数 $\times 2$

なお，RPTR は双方向リストの先頭と末尾の要素を格納するために 2 倍している．

### 3.3　動作

LSC の読み出しおよび書き込みフローを図 **4** に示す．

(1) 読み出しヒット時の動作
読み出し要求時には参照アドレスのインデックス部分に基づいてタグ・セットが選択され，そのタグ部分と参照アドレスのタグ部分を比較し，キャッシュヒットまたはキャッシュミスを判定する．ヒット時にはタグ・セットの FPTR を参照することで，適切なラインをデータアレイから読み出す．ここで，タグ比較と FPTR の参照は同時に行えるため，FPTR の参照に要する時間は隠蔽可能であり，LSC の読み出しヒット時間は従来型キャッシュの読み出しヒット時間と同じである．

(2) 読み出しミス時の動作
キャッシュミス時には置き換えアルゴリズムにより，タグ・セットから追い出しエントリが選択され，追い出し対象ラインにフィルするために主記憶からデータを読み出す．追い出しエントリのタグ部分を更新し，主記憶からのデータを入力とし図 **5** に示す LSC-Update と呼ばれる処理を行う．同時に主記憶からのデータは上位メモリ階層に転送される．なお，LSC-Update は LSC に固有な，データアレイにデータを書き込む際（書き込み命令やデータのフィル動作）に必要となる処理である．具体的には，書き込む値がデータアレイ内に存在するかの検索，またそれにともないデータに対応しているデータアレイ（RPTR）およびタグアレイの双方向リストの管理（FPTR および TLIST）などがこれにあたる．

LSC-Update の動作フローを図 5 に示す．FPTR を参照することで，データ・セットが読み出され，当該データ・セット内の各データ値は書き込みデータ値と比較される．比較結果によりデータ値ヒット，データ値ミスが判定される．

- データ値ヒット時の動作
  データ値ヒットの場合，FPTR を，ヒットしたデータ値が保持されているデータアレイ内のラインを指定するよう更新する．また，データ値を共有するタグエントリの双方向リストの末尾にエントリ

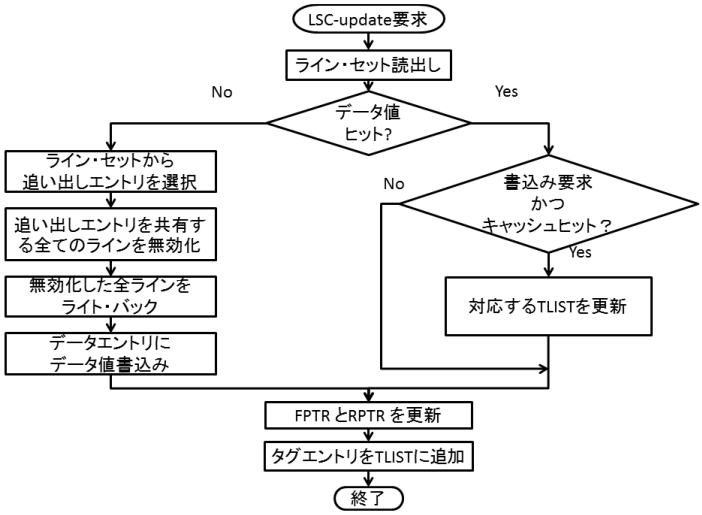


図 5　LSC-Update 動作のフロー
**Fig. 5**　Detailed flow of the LSC-Update operation.

を追加するため，データエントリの RPTR の tail エントリが更新される．タグアレイ側においても，双方向リストの末尾に該当していたタグエントリの TLIST および挿入されたエントリの TLIST を更新する．また，タグアレイにタグエントリ挿入の際にあるタグエントリを追い出し，かつそのエントリがデータ値を他のエントリと共有していた場合には，該当していた双方向リストからリストを削除するように前後のリストの TLIST が更新される．

- データ値ミス時の動作
  データ値ミスの場合，ラインの置き換えのために，追い出しエントリがデータ・セットから選択される．このとき，追い出しエントリに対応しているタグエントリはすべて無効化する必要がある．そのため，RPTR の示す先頭エントリに対応するタグエントリを無効化し，TLIST をたどって最後方のエントリまですべてのタグエントリを無効化する．同時に書き込みデータ値をデータアレイに書き込み，RPTR をタグエントリと対応付ける．また，タグエントリも書き込みデータを示すように FPTR を更新し，TLIST の前後のリストには他のタグエントリとデータを共有していない（データ値ミスのため）ことを示す NULL を書き込む．

(3) 書き込みヒット時の動作
書き込み要求時にはタグ・セットが参照され，読み出し要求時と同様にキャッシュヒットまたはキャッシュミスが判定される．キャッシュヒットの場合，タグエントリの追い出しは発生せず，データアレイでの更新のための LSC-Update 処理を行う．
ただし，書き込みデータ値が元のデータ値と変わっていた場合は新しいラインにタグが対応付けることに加え，タグと元ラインとの対応関係を切り離すためのポインタ処理が必要になる．このポインタ処理では元ラインを共有するタグエントリの双方向リストから書き込み対象のタグエントリを削除するために該当する TLIST を更新する．

(4) 書き込みミス時の動作
キャッシュミスの場合，追い出しタグエントリが選択されフィルのために主記憶からデータを読み出し，タグを更新した後に LSC-Update 処理を行う．

なお，LSC-Update では「書き込んだデータにより，既存の共有状態が破壊された場合の共有状態の解除処理」「追い出しエントリを共有する全ラインの無効化処理」において TLIST をたどるためにタグアレイを複数回検索する必要がある．この検索はすべて逐次的に行うことを想定しているため，簡単な制御回路シーケンサを追加するのみでハードウェア実装可能である．この制御回路の面積はキャッシュ全体の面積に対して無視できる程度に小さいため，ハードウェア・コストの視点から実装上解決すべき重要な課題とならない．

また，LSC では LSC-Update という従来型のキャッシュが持たない処理が追加されており，データアレイ側の書き込みのレイテンシが伸びることが懸念される．しかしながら，書き込み時のレイテンシが及ぼす影響は，実装上解決すべき重要な課題にならないと考えられる．近年のハイエンドのプロセッサには書き込みのレイテンシを隠蔽するためにライトバック・バッファが設けられており，キャッシュに対する書き込み要求実行中に発行されたアドレスとデータはこのバッファに格納される．たとえば，Intel Core 2 プロセッサのマイクロアーキテクチャを最大限模倣しているサイクルレベルシミュレータの Zesto では 8 エントリのライトバック・バッファが設けられている．このようなライトバック・バッファの搭載を前提にすることは現実的

であり，性能低下を緩和できると考えられる．ただし，書き込みレイテンシの増加によりライトバック・バッファがあふれると，プロセッサ性能に悪影響を及ぼす可能性がある．このような状況が頻繁に発生する場合には，書き込みレイテンシの削減が実装上重要な課題になる．

## 4. 評価

### 4.1 評価方法

ライン共有キャッシュの性能を評価することを目的にシミューレションを行った．プロセッサシミュレータにはM5 [1] を用い命令セットアーキテクチャはAlphaとし，プロセッサの構成は**表 1** に示すものとした．ベンチマーク・プログラムはSPEC CPU2000 [3] から12本（parser, vortex, mcf, twolf, vpr, ammp, apsi, art, applu, sixtrack, mgrid, swim）を選択した．プログラムの入力はtrainデータセットとし，プログラムの最初から最後までを実行して評価を行った．

LSCの最も興味深い特徴はデータエントリに比べ大多数のタグエントリを持つことができる点である．そこで，LSCのタグとデータの比率に**表 2** に示す4種類の構成（LSC-2，LSC-4）を用いて評価を行った．LSC-XはLSCが従来型L2キャッシュに比べX倍のタグエントリを有することを表す．タグエントリ数とデータエントリ数は従来型キャッシュのSRAMビット数を超えないように選択した．LSC-8，LSC-16の容量はタグアレイ側のビット数だけで従来型L2キャッシュの容量に対して大きくなってしまうが，性能を見積もるための参考として評価した．データアレイの連想度は性能の上限を評価するためにフル・アソシアティブによる評価と現実的な値としてデータアレイの連想度を16とした場合の評価を行う．比較対象の従来型キャッシュの構成は連想度16，容量256 KBとする．また，置き換えアルゴリズムには，タグ・セットおよびデータ・セットの置き換えにLRUを用いる．

**表 1**　シミュレーション対象のプロセッサ構成
**Table 1**　Configurations of the simulated processor.

| | |
|---|---|
| コア・アーキテクチャ | 1コア，One-IPCモデル |
| L1命令キャッシュ | 32 KB，2-way<br>64 Bライン，1 cycleレイテンシ |
| L1データキャッシュ | 32 KB，2-way<br>64 Bライン，2 cyclesレイテンシ |
| 主記憶レイテンシ | 200 cycles |
| 従来型L2キャッシュ | 256 KB，16-way<br>64 B，12 cyclesレイテンシ |

LSCを用いると読み出しヒット時のレイテンシは増加しないが読み出しミスおよび書き込み時のレイテンシが増加することがある．ここでは，LSCにより従来型キャッシュのレイテンシは増加しないと仮定してLSCのレイテンシは従来型キャッシュのレイテンシと同じとした．これは楽観的な仮定であるが評価によりLSCの性能向上の上限を調べることができる．LSC-2，LSC-4，LSC-8，LSC-16の性能およびMPKI（misses per kilo instructions）を評価した．

### 4.2 評価結果

まずはじめに，データアレイがフルアソシアティブである場合のLSCの従来型キャッシュに対する性能向上比を**図 6** に示す．$x$ 軸はベンチマーク・プログラム $y$ 軸は従来型LLCに対する性能向上比を表す．図から分かるように，`parser`，`vortex`，`sixtrack`，`mgrid`，`swim` を除いてLSC-4はLSC-2と比較して大きく性能向上する．これは，データエントリ数を4分の1に減少させてもデータ値を共有することによりタグエントリがうまく活用されていることを表す．特に `mcf`，`ammp`，`twolf`，`apsi` では大きな性能向上が得られた．最も性能向上した `mcf` では従来型キャッシュに対し35%の性能向上が得られた．LSC-8，LSC-16では `vpr`，`art` を除いてLSC-4と同様の傾向が得られたが，性能の向上幅はLSC-2に対するLSC-4ほどではない．`vpr` ではタグエントリ数を増加させるにつれLSC-2に対して性能が低下する．また，`art` ではLSC-2，LSC-4では性能向上しないがLSC-8，LSC-16ではそれぞれ34%，40%の性能向上が得られた．これらのプログラムの結果については後に考察する．

次に，従来型キャッシュとLSCのMPKIを**図 7** に示す．大きな性能向上が得られたベンチマーク・プログラム（`mcf`，`ammp`，`twolf`，`apsi`）では大きくMPKIが減少している．その他のベンチマーク・プログラムでは `vpr` を除いてMPKIはわずかに減少するないしは変化しない．`vpr` で

**表 2**　従来型L2キャッシュとライン共有キャッシュの構成
**Table 2**　Configurations of LSC and the conventional L2 cache.

| | Conv. L2 | LSC-2 | LSC-4 | LSC-8 | LSC-16 |
|---|---|---|---|---|---|
| タグエントリ数 | 4 K | 8 K | 16 K | 32 K | 64 K |
| データエントリ数 | 4 K | 2 K | 1 K | 512 | 256 |
| タグ・ポインタアレイの連想度 | N/A | 16 | 16 | 16 | 16 |
| データアレイの連想度 | N/A | 16<br>full-assoc | 16<br>full-assoc | 16<br>full-assoc | 16<br>full-assoc |
| 合計ビット数 | 288 KB | 221 KB | 240 KB | 387 KB | 722 KB |

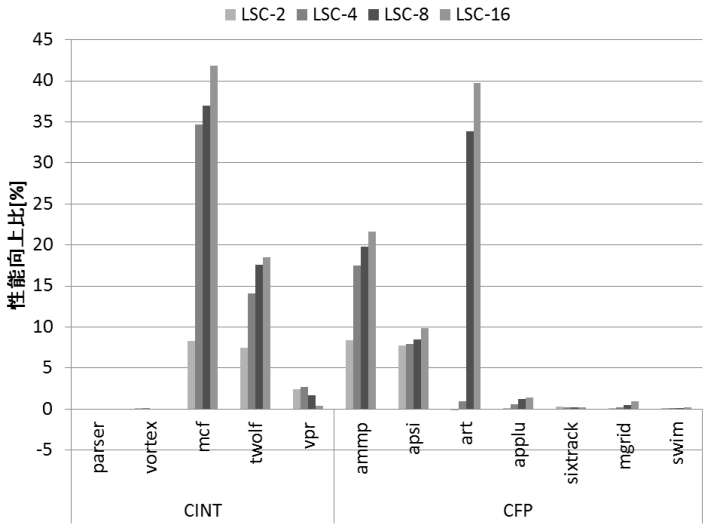


**図 6**　データアレイをフル・アソシアティブとした LSC の従来型キャッシュに対する性能向上比

**Fig. 6**　Speedup of LSC whose data array is full associative against the conventional cache.

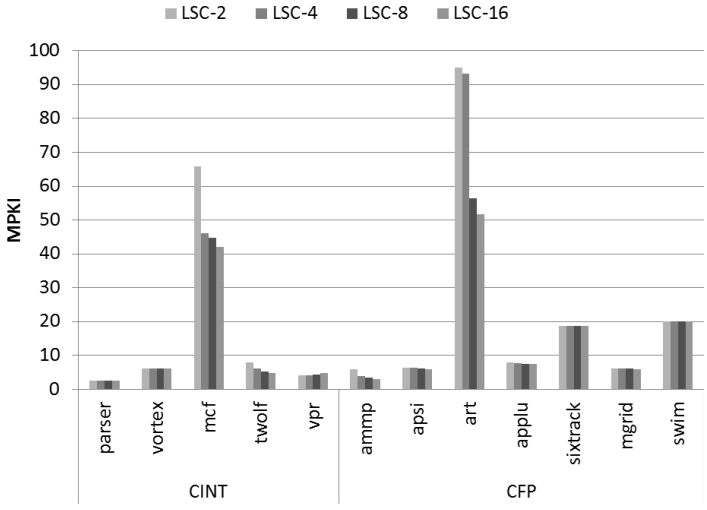


**図 7**　従来型キャッシュと LSC の MPKI

**Fig. 7**　MPKI of LSC whose data array is full associative and the conventional cache.

は LSC-8，LSC-16 においてミス回数が LSC-2 に対して増加するために LSC-2 に対して性能が低下することが分かる．また，`art` では LSC-8，LSC-16 においてミス回数が大幅に削減されたため性能が大きく向上した．

図 6 および図 7 で見たように LSC の性能向上は従来型キャッシュに比べより多くのデータを格納することで MPKI を削減することにある．MPKI がサイズを大きくした従来型キャッシュにどれだけ迫っているかを確認するために，従来型キャッシュの MPKI を容量を 256 KB，512 KB，1,024 KB，2,048 KB，4,096 KB と変化させて得られた値を**図 8** に示す．なお，512 KB，1,024 KB，2,048 KB，4,096 KB のタグエントリ数はそれぞれ LSC-2, LSC-4, LSC-8，LSC-16 のタグエントリ数と等しいことに注意されたい．また，参考のために最も性能向上が得られた LSC の構成を `LSC-best` としてその MPKI を図に示している．`LSC-best` はほとんどのベンチマーク・プログラムにおいて 1,024 KB と同等の結果である．この結果および，図 6 で示したように LSC-8 および LSC-16 の LSC-4 に対する性能向上は `art` を除いて小さく，`LSC-4` は性能とハードウェアオーバヘッドの観点から優れていることが分かる．キャッシュ容量を 256 KB から 1,024 KB に増加させることで MPKI が大幅に低下するプログラム（mcf，ammp，twolf，apsi）では図 6 で大きな性能向上が得られる．これは LSC がデータエントリ数の削減による影響を受けずに増加したタグエントリを効率的に利用できていることを意味する．そのためには，データエントリ数の減少による影響を緩和するために高いデータ値の局所性，つまり低い一意データ存在率が必要となる．

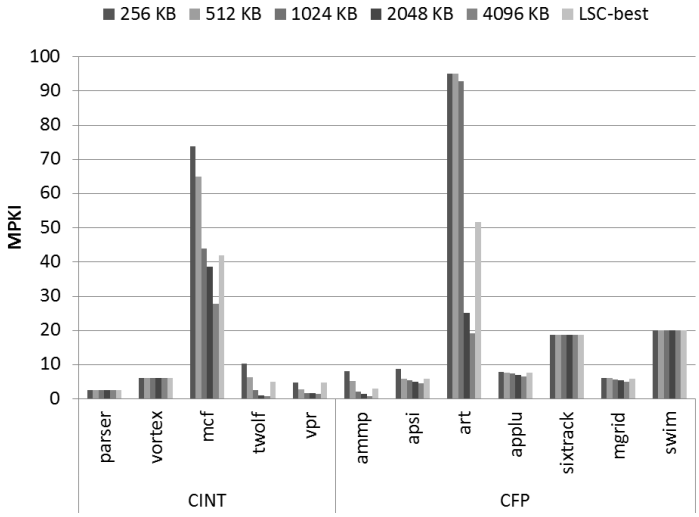


図 8　異なるキャッシュ容量に対する MPKI
**Fig. 8**　MPKI of different LLC size.

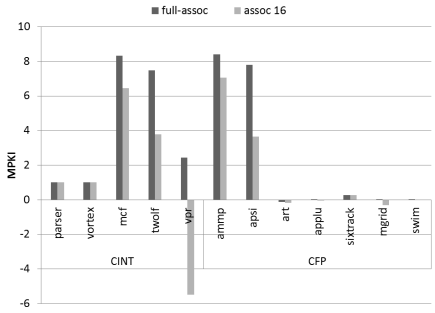


図 9　LSC-2 において異なるデータアレイの連想度に対する性能向上比の比較
**Fig. 9**　Speedup of LSC-2 of different associativity of data array.

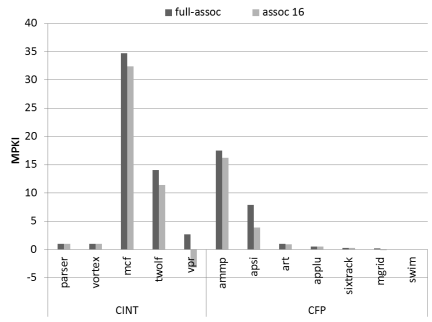


図 10　LSC-4 において異なるデータアレイの連想度に対する性能向上比の比較
**Fig. 10**　Speedup of LSC-4 of different associativity of data array.

続いて，データアレイの連想度が性能に与える影響を考察する．**図 9** および**図 10** に LSC-2 および LSC-4 においてデータアレイの連想度を変化させたときの従来型のキャッシュに対する性能向上比の結果を示す．full-assoc はデータアレイの連想度がフルアソシアティブである場合を，assoc-16 はデータアレイの連想度が 16 である場合の性能向上比を表す．LSC-4 においてはほとんどすべてのベンチマークで assoc-16 が full-assoc に近い性能を達成しているのに比べて，LSC-2 では `mcf`，`ammp`，`twolf`，`apsi`，`vpr` において大幅に性能が低下している．これは，これらのベンチマークにおいてはデータアレイ側の追い出しによって無効化されたタグが再び参照されてキャッシュミスとなることが多く起こるため full-assoc に比べ assoc-16 の性能が大きく低下したと考えられる．一方で LSC-4 においてはデータアレイ側のコンフリクトミスがほとんど生じなくなるのに十分なセット数を備えていることが分かる．

また，`vpr` において LSC-2 と LSC-4 の assoc-16 はタグエントリ数がそれぞれ 2 倍，4 倍となっているにもかかわらず従来型キャッシュよりも性能が低下している．この理由を調べるために，従来型キャッシュと LSC-1 の assoc-16 の MPKI を**図 11** に示す．LSC-1 は従来型キャッシュと同数のタグエントリとデータエントリを有する LSC である．従来型キャッシュとの違いは 3.3 節の「データ値ミス時の動作」で述べているように，追い出されたラインを共有するすべてのタグの無効化が発生する点にある．このことにより無効化されるタグエントリがさらにキャッシュミスを引き起こす．図より，`vpr` を除くベンチマークでは大幅な MPKI の増加は確認できない．一方 `vpr` では従来型キャッシュの MPKI が 4.9 なのに対し LSC-1 では 7.9 と大幅に増加している．このことから `vpr` はデータエントリ側のコンフリクトミスが原因でキャッシュミスが増加し，LSC-2 と LSC-4 でも従来型のキャッシュに対して性能が低下してしまったといえる．

## 5.　関連研究

これまでにも，データ値の局所性に着目したキャッシュ構成法に関するいくつかの研究が行われている [6], [7], [8]．文献 [8] では，データ圧縮を応用した L1 キャッシュとして Compession Cache（CC）を提案している．具体的には，

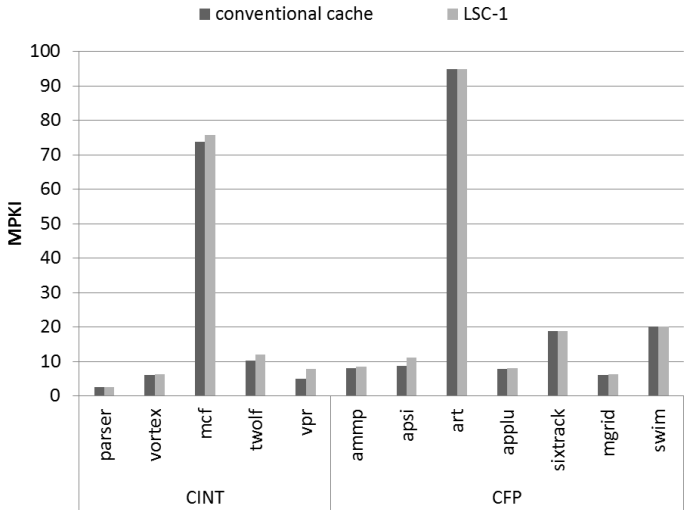


図 11　従来型キャッシュと LSC-1 における MPKI の比較
**Fig. 11**　MPKI of different LLC size.

1 個のラインに非圧縮ラインを 1 個，または，半分以下のサイズに圧縮したラインを 2 個格納する．この圧縮において，データ値の局所性を利用する（頻繁に参照されるデータ値を圧縮する）．文献 [7] では，低消費電力を実現する L1 キャッシュとして Frequent Value Cache（FVC）が考案されいる．FVC では頻繁に参照される値を数ビットに圧縮し，それ以外の値を圧縮せずに格納する．文献 [6] では，キャッシュ内に多く存在するデータ値 0 を利用する Dynamic Zero Compression（DZC）を提案している．通常，圧縮対象となるデータ値の数が増えると圧縮および復元のためのオーバヘッドが増加し，ひいてはキャッシュ・アクセス時間の増大を招く恐れがある．この問題を回避するため，これらの関連研究では圧縮対象とするデータ値の数を制限している．これに対し，本稿で提案する LSC は，圧縮対象とするデータ値の数に制限がないという利点がある．LSC では，低レイテンシ化よりもキャッシュミス率の削減に重きをおく LLC への適用が前提となり，より多くのデータ値をキャッシュ内に格納することができる．

NuRAPID [2] と V-Way Cache [5] は，データの配置場所とタグの配置場所が分離されているという点で LSC に類似している．V-Way Cache [5] は，キャッシュ内の各セットの連想度をプログラム要求に応じて変更することができる．データエントリ数に比べ多数のタグエントリを有するキャッシュ構成においてタグとデータの配置場所を分離することでこれを可能にしている．NuRAPID [2] はタグとデータの連続アクセスを利用し，タグとデータの配置場所を分離することで頻繁にアクセスされるデータを多く最も高速なサブアレイに配置することを可能にする．これらに対し，LSC はデータ値の局所性を利用して複数のタグを単一のデータに関連付けることを目的としている点が本質的に異なっている．

## 6. おわりに

本稿では，新しいキャッシュ・アーキテクチャとしてライン共有キャッシュ（LSC）を提案した．LSC では，プログラムの実行において多くのキャッシュラインが同一のデータ値を有するという特性に着目し，複数タグによる単一ラインの共有を可能にする．提案方式による性能向上の上限を明らかにすることを目的とし，ベンチマーク・プログラムを用いた定量的評価を行った．その結果，従来型キャッシュと比較して最大で 35%，平均で 1.63%の性能向上を達成することができた．

LSC は従来のキャッシュ構成法とはまったくことなるアーキテクチャを採る．そこで，本研究では，LSC の潜在能力を明らかにすることを目的とし，キャッシュ・アクセス時間に関しては従来型キャッシュと同じであると仮定した．3.3 節で説明したように，読み出しレイテンシは従来型キャッシュとほぼ同程度と考えることができる．しかしながら，書き込みレイテンシの増加，ならびに，LSC-Update 処理にともなうオーバヘッドは本評価では考慮していない．今後，これらの影響を含めた性能評価を実施する予定である．また，本実験では，使用したベンチマークのワーキングセットサイズを考慮して 256 KB といった比較的小さな LLC を想定した評価を実施した．したがって，より大きなデータを処理対象とし，かつ，より大容量な LLC を前提とした評価も必要である．一方，LSC のさらなる改善としては，キャッシュラインをサブライン（たとえば，ラインサイズが 8 ワードの場合において，2 ワードや 4 ワードといった単位）に分割し，これらを 1 データエントリとして取り扱う方法が考えられる．ライン単位と比較して，サブライン単位の方が同一データ存在率が低くなる傾向にあり，より積極的なキャッシュラインの共有を期待できる．また，LSC ではデータ値の局所性が高く，ワーキング

セットサイズが大きなベンチマークでは性能向上を得られるが，データ値の局所性が高く，ワーキングセットサイズが大きなベンチマークでは性能低下を引き起こすと考えられる．これは，データエントリの不足によるデータ値ミスが頻発するためである．そこで，LSC で増加したタグ・アレイの一部をデータ・アレイに切り替えることを可能にする構成が考えられる．その結果，いずれのタイプのベンチマークにも対応可能となり，一意データ存在率が高くワーキングセットサイズが大きなベンチマークにおいて性能低下を引き起こさないことが期待できる．

### 参考文献


[1] Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A. and Reinhardt, S.: The M5 simulator: Modeling networked systems, *IEEE, Micro*, Vol.26, No.4, pp.52–60 (2006).

[2] Chishti, Z., Powell, M. and Vijaykumar, T.: Distance associativity for high-performance energy-efficient non-uniform cache architectures, *Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003, MICRO-36*, pp.55–66, IEEE (2003).

[3] Henning, J.: SPEC CPU2000: measuring CPU performance in the New Millennium, *Computer*, Vol.33, No.7, pp.28–35 (online), DOI: 10.1109/2.869367 (2000).

[4] Kurd, N., Mosalikanti, P., Neidengard, M., Douglas, J. and Kumar, R.: Next Generation Intel¯ Core? Micro-Architecture (Nehalem) Clocking, *IEEE Journal of Solid-State Circuits*, Vol.44, No.4, pp.1121–1129 (2009).

[5] Qureshi, M., Thompson, D. and Patt, Y.: The v-way cache: Demand-based associativity via global replacement, *Proc. 32nd International Symposium on Computer Architecture, 2005, ISCA'05*, pp.544–555, IEEE (2005).

[6] Villa, L., Zhang, M. and Asanović, K.: Dynamic zero compression for cache energy reduction, *Proc. 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pp.214–220, ACM (2000).

[7] Yang, J. and Gupta, R.: Frequent value locality and its applications, *ACM Transactions on Embedded Computing Systems* (*TECS*), Vol.1, No.1, pp.79–105 (2002).

[8] Yang, J., Zhang, Y. and Gupta, R.: Frequent value compression in data caches, *Proc. 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pp.258–265, ACM (2000).



**岡 慶太郎**

昭和 63 年生．平成 23 年九州大学工学部電気情報工学科卒業．平成 23 年同大学大学院修士課程に進学，現在に至る．ダイナミック・マルチコアに関する研究に従事．



**阿部 祐希**

昭和 63 年生．平成 23 年九州大学工学部電気情報工学科卒業．平成 23 年同大学大学院修士課程に進学，現在に至る．キャッシュメモリ，GPU に関する研究に従事．



**佐々木 広**　（正会員）

2003 年東京大学工学部計数工学科卒業．2005 年同大学大学院情報理工学系研究科修士課程修了．2008 年同大学院工学系研究科博士課程修了．博士（工学）．同年東京大学先端科学技術研究センター特任助教，2010 年より東京大学大学院情報理工学系研究科特任助教を経て，現在，九州大学大学院システム情報科学研究院特任准教授．計算機アーキテクチャ，オペレーティングシステムの研究に従事．IEEE，ACM，USENIX 各会員．



**井上 弘士**　（正会員）

昭和 46 年生．平成 8 年九州工業大学大学院情報工学研究科修士課程修了．同年横河電機（株）入社．平成 9 年より（財）九州システム情報技術研究所研究助手．平成 11 年の 1 年間 Halo LSI Design & Device Technology, Inc. において訪問研究員としてフラッシュ・メモリの開発に従事．平成 13 年九州大学において工学博士を取得．同年福岡大学工学部電子情報工学科助手．平成 16 年より九州大学大学院システム情報科学研究院助教授．平成 19 年 4 月より同大学准教授，現在に至る．高性能/低消費電力プロセッサ/メモリ・アーキテクチャ，ディペンダブル・アーキテクチャ，3 次元積層アーキテクチャ，性能評価，等に関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．

**村上 和彰**　（正会員）

昭和 35 年生．昭和 59 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．同年富士通（株）入社．汎用大型計算機の研究開発に従事．昭和 62 年九州大学助手．平成 6 年九州大学助教授，情報基盤研究開発センター長，情報統括本部長．現在，九州大学大学院システム情報科学研究院情報知能工学部門教授．計算機アーキテクチャ，並列処理，システム LSI 設計技術，等に関する研究に従事．工学博士．平成 3 年情報処理学会研究賞，平成 4 年情報処理学会論文賞，平成 9 年坂井記念特別賞，平成 12 年日経 BP 社 IP アワード，平成 12 年情報処理学会創立 40 周年記念論文賞，平成 14 年電子情報通信学会業績賞をそれぞれ受賞．