情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 3　1–11 (Sep. 2009)

部分的試行に基づく動的共有キャッシュ分割方式

小川周吾[†1]　入江英嗣[†1]　平木敬[†1]

マルチコア CPU における共有キャッシュでは，コア間の競合によるミスが発生する．その回避手段として各コアの使用領域を制限する動的キャッシュ分割が提案されている．しかし従来のキャッシュ分割方式では，LRU 以外の置換方式への対応が難しい点，共有キャッシュのミス最小化に多くのハードウェア資源を要する点が問題である．本論文では任意の置換方式を用いる共有キャッシュに対してミスを低減させるための，部分的試行に基づく動的共有キャッシュ分割方式（PTRP: Partial Trial-Based Runtime Partitioning）を提案する．PTRP ではキャッシュの少量のセットを計測セットと定め，試験的にキャッシュの分割を変更してミスの増減を調べる．これにより任意の置換方式に対応し，少量のハードウェア追加でミス数を低減するキャッシュの分割を発見できる．PTRP を性能およびハードウェア資源量で評価した結果，キャッシュを均等に分割した場合と比べて 2 コアで最大 31%，4 コアで最大 19%のミス削減効果を確認した．また従来方式と比べて 2 コアの場合に平均 1.5%，4 コアの場合に平均 3%のミス増加に対し，追加ハードウェア容量を約 12%まで削減した．

Partial Trial-based Runtime Partitioning Method for Shared Caches

Shugo Ogawa,[†1] Hidetsugu Irie[†1] and Kei Hiraki[†1]

To avoid the increase of conflict misses among cores which share cache on multicore CPU, many dynamic cache partitioning methods have been proposed. However these methods have two problems. First, those methods can only manage shared caches which depend on LRU replacement. Second, those methods require many resources to reduce cache misses. In this paper, we propose Partial Trial-Based Runtime Partitioning Method (PTRP) for reducing cache misses in shared caches through dynamic cache partitioning. PTRP uses a few sets called “monitoring sets” for applying new cache partition and monitoring cache misses as a trial. This trial helps finding a new partition for reducing cache misses with a little additional hardware. We evaluate an effect for performance and resource of PTRP, then we show that PTRP reduces 31% misses in 2-core CPU cache, 19% misses in 4-core CPU at the maximum. We also show that PTRP reduces hardware requirement to 12% with 1.5% cache miss increase in 2-core CPU, 3% miss increase in 4-core CPU on the average.

1. はじめに

近年普及しているマルチコア CPU では各コア上で同時に複数プロセスの実行が可能である．マルチコア CPU では同時実行プロセス数の増加にともない，CPU 全体のメモリアクセス数が増加する．そのためマルチコア CPU ではシングルコア CPU と比べて，性能低下を回避するためにキャッシュミスの削減によるメモリアクセス回数の削減がより重要である．一般にマルチコア CPU では，メモリに近い低次かつ大容量の set-associative キャッシュを複数コア間で共有する．コア間でキャッシュを共有することで，単一コアが大容量のキャッシュを利用できる．また，各コアに大容量のキャッシュを実装するより資源面で有利である．

一方でマルチコア CPU の共有キャッシュでは，コア間のアクセス競合によりミスが発生する．競合によりキャッシュ上の頻繁に参照されるデータが参照頻度の低いデータで置換されることでミスが増加する．また競合によるキャッシュミスは複数コア間でも発生する．さらにコア間の競合によるミスの発生頻度は，CPU 内のコア数増加に比例して増加する．

これらのプロセス間での競合ミス発生を避けるために各コアに対する動的キャッシュ分割方式が提案されている．キャッシュ分割では共有キャッシュ内を各コアの占有可能な領域に分割することで，プロセス間の競合を防止する．動的キャッシュ分割では，キャッシュの省電力化[1)–3)]，公平性[4)–6)]，QoS 管理[7)] を目的とした方式が提案されている．またキャッシュ全体のミス数の最小化を目的として各コアのキャッシュ占有量を動的に変更する方式が提案されている[8)–10)]．これらの方式では，各コアの占有キャッシュ量に対するミス数を調べる．その結果を用いてキャッシュ分割を行い各コアのキャッシュ占有量を指定する．

しかしこれらの動的キャッシュ分割方式は，LRU 以外の置換方式のキャッシュでは利用不可能な点が課題である．これは動的キャッシュ分割方式が LRU による置換の性質を用いて各コアの占有 way 数に対するミス数を計測するためである．加えてミス数の情報取得に多くのハードウェア資源が必要な点も課題である．

本論文では任意の置換方式を用いるキャッシュに適用可能な部分的試行に基づく動的共有

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

キャッシュ分割方式（PTRP）を提案する．PTRP はキャッシュの一部セットに対して他の領域とは異なる分割を試行することで，キャッシュ分割に対するミス数の増減を調べる．試行の結果ミス数を低減する分割のみをキャッシュ全体に適用する．PTRP はキャッシュ分割の試行を繰り返すことで，少ない資源量でミスを最小化するキャッシュ分割を行う．

2. 関 連 研 究

2.1 ミス最小化を目的とした共有キャッシュ分割

Stone ら[11] は複数プロセスが利用するキャッシュのミスが最小となるように way 単位で分割を行う方式を提案した．Stone らの研究では各プロセスに対して割り当てられたキャッシュの way 数とミス数の関係を静的に調べた結果を用いて，キャッシュ全体のミスが最小となる way の分割数を求める greedy アルゴリズムを用いた．しかし Stone らの方式では，事前実行により共有キャッシュの割当て way 数変更時のミス回数の記録が必要である．

Suh ら[8),9)] はマルチコア CPU 上の共有キャッシュのミスを最小化するために，動的な共有キャッシュの way 分割方式を提案した．この方式では各プロセスの使用するキャッシュについて 1 way あたりのミス減少数である “marginal gain” を定義して，この値が最大値となる way 分割数を探索することでミスを低減させる．また Stone らの greedy アルゴリズムを改良して，新たに non-convex と呼ばれる，プロセスの占有 way 増加時のミス数の変化が単調減少ではない場合に対応している（**図 1**）．しかし Suh らの提案方式は，CPU 上の各コアに対して way ごとの marginal gain を調べるために多くのハードウェア資源を必要とする．

Qureshi ら[10] は，キャッシュと別に新たに追加した Utility Monitor（UMON）を用いた marginal gain （Qureshi らは “marginal utility” と定義）の取得方式，および動的キャッシュ分割方式として Utility-Based Cache Partitioning（UCP）を提案した．UMON はコアごとに実装され，way ごとのヒットカウンタ，ヒットの判定に用いるタグ情報を持つ．また Suh らのアルゴリズムより少ない計算量で non-convex なミス数変化に対応し，キャッシュミスを最小化する way 分割を求める lookahead アルゴリズムを提案している．さらにキャッシュアクセス数，ヒット数の計測機能を一部のセットに限定する Dynamic Set Sampling（DSS）[12] を用いることで，Suh らの方式と比べてハードウェア資源量を削減している．DSS ではセット数 1024 のキャッシュに対して 32 セットの計測で十分な精度が得られることを示している．

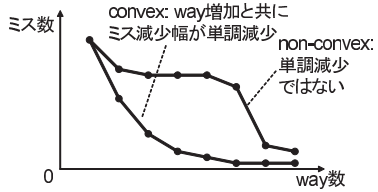

図 1　キャッシュの way 数増加に対する non-convex なミス数減少
Fig. 1　“Non-convex” miss decrease according to cache way increase.

以上の関連研究では，コアごとにとりうる占有 way 数すべてに対するミス数を調べることでミスを最小化するようにキャッシュ分割を行う点が共通する．一方 Dybdahl ら[13] は，キャッシュ分割による各コアの割当て way 数を 1 増減した場合のミス数を用いて，キャッシュミスが低減するように分割 way 数を段階的に変更する Cache-Partitioning Aware Replacement Policy（CPAR）を提案した．この方式はキャッシュの各セットにコアごとに，直前にキャッシュから置換されたデータへのアクセスを調べる shadow tag を追加する．同時に LRU ブロックのヒット回数を計測して，各コアの占有 way 数を 1 増減した場合のミス数を算出する．

2.2 従来の共有キャッシュ分割方式の課題

共有キャッシュのミス削減を目的とする動的キャッシュ分割方式[9),10)] は，以下の 2 段階の処理を行う．まず各コアについて占有 way 数ごとにキャッシュミス数を計測する．各コアの任意の占有 way 数に対するミス数の計測を，占有 way 数を変更せずに同時に行う．つまり一般の CPU の性能カウンタではミス数を計測できない．次に計測された各コアの占有 way 数ごとのキャッシュミス数から，キャッシュ全体のミス数を最小化する way 分割を決定する．

従来方式では way 分割の決定に必要なキャッシュミス数計測のためにコアごとに共有キャッシュの way 数と同数のカウンタを使用する（**図 2**）．各カウンタは共有キャッシュでヒットしたブロックの，MRU から数えた順番に対応する．たとえばヒットしたブロックの MRU から数えた順番を n とする場合，n 番目でヒットした回数を記録するカウンタの値を更新する．

カウンタ値から各コアの任意の占有 way 数に対するキャッシュミス数を求める．キャッシュミス数の算出には，LRU による置換で成立する stack property[14] を利用する．LRU で置換を行うキャッシュでは，way 数 n でヒットするアクセスは，way 数が n 以上であれば必ずヒットする．たとえば way 数が n の場合のキャッシュミス数 $M(n)$ は，総アクセス数を N，MRU から数えて i 番目のブロックのヒット数を $H(i)$ とすると以下の式 (1) のとおり求められる．

$$M(n) = N - \sum_{i=1}^{n} H(i) \quad (1)$$

一方従来方式のうち CPAR[13] では，LRU のブロックと，キャッシュから置換された直後のブロックのアクセス数を計測する．CPAR でも stack property を利用して，計測したヒット数から占有 way 数が 1 増減した場合のミス数の増減を求める．

したがってこれらの動的キャッシュ分割方式は，LRU 以外の置換方式を用いたキャッシュへの適用が困難である点が第 1 の課題である．たとえば LRU の代替方式である疑似 LRU は，LRU が持つ各セット内のブロックアクセスの順序情報を簡略化することで実装を容易にした置換方式である．疑似 LRU では順序情報が簡略化されるため，LRU とブロックの置換順序が異なる．つまり stack property は成立せず，各 way 数に対するミス数の計測は困難である．また stack property の成立しない他の置換方式も同様にミス数の計測が困難である．

第 2 の課題は，CPU にヒット数計測用に多くのハードウェア資源が必要な点である．たとえば各コアに対して共有キャッシュの全 way 数分のタグ情報が必要である．このタグ情報は各コアが実際より多い way 数を占有した場合のヒットを判定するために使用する．そのため必要ハードウェア資源が CPU コア数，共有キャッシュの way 数に比例する．一方従来方式におけるハードウェア資源の削減方式である DSS[12] は，ヒット数の計測をキャッシュ内の一部のセットに限定することでタグを削減する．しかし DSS を用いた場合は，タグ情報，カウンタの必要数が CPU のコア数とキャッシュの way 数に比例する．

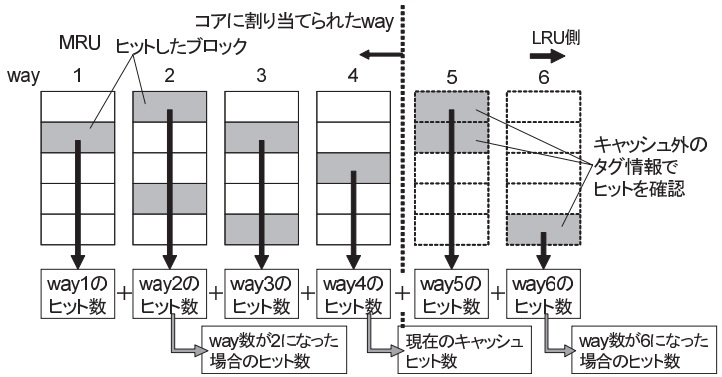

図 2　キャッシュの MRU から LRU までの way とヒットカウンタ
Fig. 2　Cache blocks and hit counters from MRU way to LRU way.

3. 部分的試行に基づくキャッシュ分割（PTRP）

本章では部分的試行に基づく動的キャッシュ分割方式（PTRP）によるマルチコア CPU 全体のキャッシュミス数削減方式を提案する．本節では PTRP のキャッシュ構成，最適キャッシュ分割の探索，PTRP の実装の順に述べる．

3.1　PTRP のキャッシュ構成

マルチコア CPU における共有キャッシュ分割では，各コアに対するキャッシュ分割量を決定する必要がある．CPU 全体のキャッシュミスの最小化が目的の場合，各コアで発生するミス数の総和が最小化されるように分割 way 数を決定する．この分割 way 数は，各コアの任意の占有 way 数に対するミス数から算出できる．しかし LRU 以外のキャッシュ置換が行われる場合，前述のとおり各コアの占有 way 数に対するミス数の正確な計測は困難である．CPU 全体のキャッシュミス数を最小化する分割は実際に分割 way 数を変更後，ミス数を計測して求めることが可能である．

PTRP では最適分割 way 数決定のために，キャッシュの一部セットのみに対して分割 way 数を変更して分割 way 数変更後のキャッシュミス率を計測する．計測結果から最適な分割 way 数を推定後，キャッシュ全体を最適 way 数で分割する．つまり PTRP では部分的試行として，共有キャッシュの一部のみに異なる way 数での分割を適用する機能を追加する．

PTRP におけるキャッシュ構成を**図 3** に示す．共有キャッシュから静的かつ無作為に選択

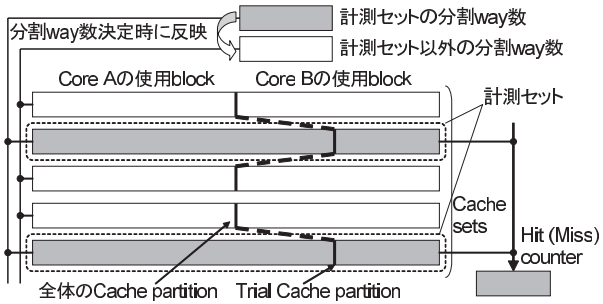

図 3　PTRP のキャッシュ構成
Fig. 3　CPU cache structure for PTRP implementation.

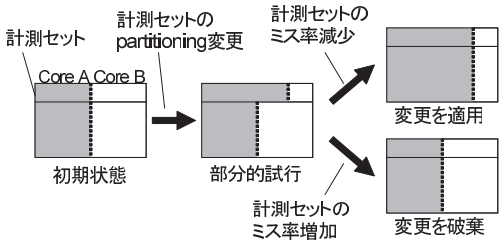

図 4　部分的試行を利用した分割 way 数決定
Fig. 4　Determination of cache way partitioning using partial-trial.

したセットを計測セットと定義して分割 way 数決定に使用する．計測セットはキャッシュ全体とは独立した分割 way 数の設定機能を持つ．また前述の DSS と同様に，キャッシュの一部領域である計測セットのみに対するアクセス数，ヒット数の計測機能を持つ．これらの機能により，試験的に分割 way 数を変更してキャッシュミス率を計測する．直接分割 way 数を変更して計測するため必要なハードウェア量は従来方式に比べて少量である．

PTRP による分割 way 数決定は，実際に分割 way 数を変更してキャッシュミス率を計測した場合に発生しうる性能低下の影響を低減する．分割 way 数の変更は先に計測セットのみに行う．そのため分割 way 数変更によるキャッシュミス増加の影響は，計測セットのみに限定される．つまり部分的試行は，キャッシュの way 分割を変更してミスが減らなかった場合に，way 分割を元に戻すまでミス率が増加して性能低下が発生する[6] ことを防ぐ．

3.2　キャッシュミスを低減する分割の探索

3.2.1　最適 way 分割の探索アルゴリズム

PTRP では実際に計測セットの分割 way 数を変更して各コアの各占有 way 数に対するキャッシュミス率を計測して最適な分割 way 数を探索する．そのため分割 way 数の決定に計測が複数回必要である．部分的試行による way 分割の決定手順を**図 4** に示す．まず共有キャッシュの計測セットの分割 way 数を変更して計測セットのミス率を計測する．その結果計測セットでミス率が減少した場合のみ，way 分割を共有キャッシュ全体に適用する．

続いて上記の分割 way 数決定手順を用いた最適 way 分割の探索手順を述べる．PTRP では部分的試行を複数回行い，各コアの分割 way 数増減時のキャッシュミス率の履歴を作成する（**図 5**）．複数回の部分的試行では，各コアの現在の分割 way 数と現在の分割 way 数を 1 増減した場合についてキャッシュミス率を計測する．1 回の計測期間を試行期間と定義する．

分割 way 数ごとのミス率の履歴は，図 5 に示したとおり全コアで計測を同時に行う．まず最初の試行期間で全コアの計測セットの way 分割を現在の共有キャッシュの way 分割に揃えて現在の way 分割でのミス率を計測する．次の試行期間では各コアの計測セットの分割 way 数を全体の半数のコアで 1 増やし，残りの半数のコアで 1 減らした状態で部分的試行を適用する．さらに次の試行期間では，前回の試行で現在の way+1 で計測したコアでは現在の way−1 で，前回の試行で現在の way−1 で計測したコアでは現在の way+1 で計測を行う．以上の 3 回の計測により，任意のコア数の CPU で各コアの現在の分割 way 数と現在の分割 way 数を 1 増減した場合のキャッシュミス率の履歴が得られる．また 3 回の計測を繰り返し実行することで，各試行期間でミス率の履歴を更新する．

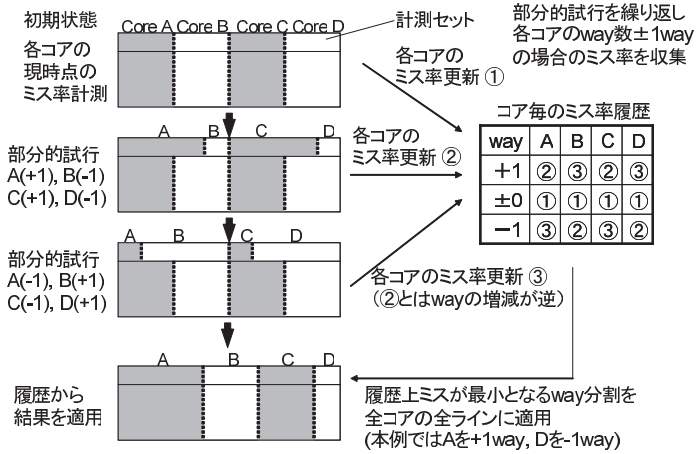

図 5　部分的試行におけるキャッシュミス率の履歴の利用
Fig. 5　Utilization of cache miss rate history information for partial-trial.

履歴が揃うと，PTRP では履歴と各コアのアクセス数から，ミス数が最小となる way 分割を求めて，共有キャッシュ全体に適用する．PTRP では分割 way 数を 1 ずつ増減する．そのため前述の各コアの way 数ごとのミス率の履歴から最適な way 分割を計算できる．分割 way 数変更後は，変更後の way 分割を基準として前述の部分的試行の繰り返しでコアごとのミス率の履歴作成と，履歴を用いた最適 way 分割の適用を行う．

3.2.2　最適 way 分割発見の時間コスト

PTRP の最適 way 分割発見の時間コストを UCP[10]，CPAR[13] と比較する．PTRP では各コアの分割 way 数を 1 ずつ変更する．そのため最適 way 分割までに現在の way 分割と

表 1　現在の way 数と次の分割 way 数変更に必要な履歴
Table 1　Relation between current partitioning and miss rate history needed for next partitioning.

	使用する履歴の way 数			
現在の way 数	$n-1$	n	$n+1$	$n+2$
n	○	○	○	
$n+1$		○	○	○

の差に比例した時間を要する．つまりコア数 N の CPU でコア n の分割 way 数が w_n，最適分割 way 数が W_n の場合最適 way 分割に必要な分割変更回数 M は以下の式 (2) で表される．

$$M = \max(|W_1 - w_1|, \cdots, |W_N - w_N|) \tag{2}$$

また PTRP では各回の分割 way 数の決定に各コアのミス数の履歴を用いる．そのため分割 way 数を 1 回変更した後に，次に分割 way 数を変更するためには，**表 1** に示したとおり新たな履歴が必要になる．たとえばコアの way 数が n の場合，PTRP では $n-1$ から $n+1$ way の場合のミス率の履歴を分割 way 数の決定に使用する．しかしコアの way 数が $n+1$ に変化すると，分割 way 数の決定に $n-1$ way の履歴の代わりに新たに $n+2$ way のミス率の履歴を使用するため，$n+2$ way について部分的試行が必要である．

PTRP では 3 回の試行期間でミス率を得るため，分割 way 数変更後に次の分割 way 数決定に必要な履歴が得られるまでに最小で 1 回の試行期間，最大で 3 回の試行期間が必要になる．また way 分割は各試行期間後に必要な履歴が揃っていれば変更できる．よって試行期間の時間を t と表すと，最適 way 分割発見までの時間 T は式 (2) で求められた，最適 way 分割発見までの変更回数 M を用いて以下の式 (3) で表される．

$$M \cdot t \leq T \leq M \cdot 3t \tag{3}$$

つまり PTRP の最適 way 分割発見までの時間は CPU のコア数に依存しない．しかし各試行期間のミス率計測では，低頻度の例外的な処理によるノイズの発生により最適 way 分割の発見が妨げられる．この場合発見に式 (3) より多くの時間を要する可能性がある．

一方 UCP では，各コアのすべての way 数でのミス数を同時に計測するハードウェアを持つ．そのため計測は 1 回で完了する．また 1 回の way 分割変更で変更可能な way に制限がない．つまり 1 回の計測時間を t とすると，最適 way 分割発見までの時間は t であり，PTRP より短い．また CPAR では，PTRP と同様に 1 回の way 分割変更では，各コアの分割 way 数の増減を 1 ずつ行う．つまり PTRP と同様に最適 way 分割発見までに式 (2) で示した回数の way 分割変更が必要である．しかし PTRP とは異なり，way 分割の変更に用いるミス数は 1 回で計測できる．つまり最適 way 分割発見までの変更回数 M と 1 回の計測時間 t を用いて最適 way 分割発見までの時間は $M \cdot t$ で表され，PTRP より短い．

3.2.3　PTRP のソフトウェアオーバヘッド

PTRP は 1 回の way 分割で使用するミス数の計測を 3 回の試行期間で行う．各試行期間では計測セットのミス回数を計測する．しかし各試行期間での総アクセス回数は異なるため，異なる試行期間で計測したキャッシュミス回数どうしの評価は不可能である．そこで PTRP はキャッシュアクセス回数をあわせて計測して，各試行期間におけるミス率を評価に使用する．

PTRP では全処理をハードウェアで実装するのではなく，試行期間の終了時にソフトウェアによる計測結果の集計，キャッシュミス率の計算，最適 way 分割の決定処理を行う．これらの処理をソフトウェアで行うことで，PTRP で使用する演算器，制御回路実装によるハードウェア増加の抑制が可能である．一方で PTRP では最適 way 分割発見に必要なソフトウェア処理を試行期間の終了ごとに実行する．そのため PTRP は試行期間の終了ごとに一定量のソフトウェアオーバヘッドを要する．つまり PTRP ではハードウェアの増加，および試行期間終了時に生じるソフトウェアオーバヘッドとの間でのトレードオフが必要である．

3.3　PTRP の実装

3.3.1　共有キャッシュの分割機能

PTRP では従来の共有キャッシュの動的分割方式[9),10)] と同様に置換方式の修正によりキャッシュ分割を実現する．修正後の置換方式ではキャッシュのセット内で各コアの占有ブロック数が各コアの way 分割数以下となるように置換を行い，共有キャッシュを分割する（**図 6**）．

修正後の置換方式ではキャッシュの各ブロックを更新したコアの番号を記録して，セットごとに各コアの占有ブロック数を調べる．キャッシュミスが発生したコアを A とすると，置換を行うセット内でコア A の占有ブロック数が A に対する分割 way 数以上の場合は，A の占有ブロックから置換先を選択する．逆にコア A の占有ブロック数が A に対する分割 way 数未満の場合は，A の占有ブロック以外から置換先を選択する．

3.3.2　キャッシュミス率の履歴集計

PTRP では 1 回の最適 way 分割の決定に必要なキャッシュミス率の履歴を 3 回の試行期間に分けて取得する．しかし各回の処理対象データ，処理内容の違いから，キャッシュミス率に差が生じる．そのため各コアが同一処理を継続する場合でも，最適 way 分割の判定が

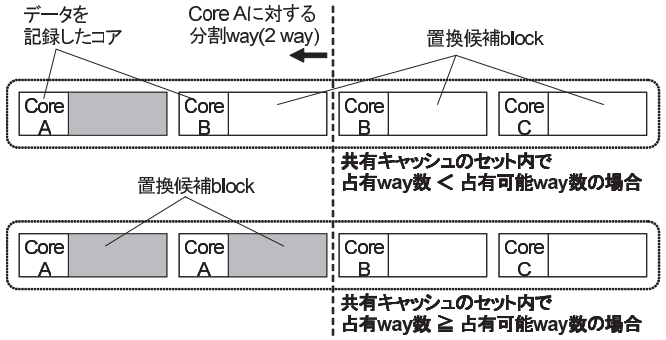

図 6　共有キャッシュ分割機能の実装
Fig. 6　Implementation of the shared cache partitioning function.

毎回変動することが問題である．

そこで実装では，各ミス率の履歴を直前の計測結果と既存の履歴との平均値とすることで問題を解決する．これにより毎回のミス率の変動を抑制する．しかし一方で，履歴の更新回数の増加にともない新規に計測したミス率が履歴に反映されにくくなる．そこで実装では，既存のミス率の履歴を R，新規に計測したミス率が r の場合，新しい履歴 R_{new} を以下の式 (4) で求める．以下本論文では $\alpha = 1$ を使用する．

$$R_{new} = (\alpha R + r) / (1 + \alpha) \ (0 \leq \alpha) \tag{4}$$

4. 性能評価

提案した PTRP と従来の共有キャッシュ全体のミスを最小化するキャッシュ分割方式の性能をシミュレータ上の実装で評価する．

4.1 評価環境

評価ではマルチコア CPU の共有キャッシュを LRU で置換した場合（LRU），コアごとに way を等分した場合（EQ），共有キャッシュのミス最小化が目的の UCP，CPAR，および提案した PTRP で動的に分割した場合の性能をシミュレーションで比較する．性能の指標には共有キャッシュ全体のミス率，および IPC を用いる．

シミュレーションパラメータを述べる．CPU およびキャッシュは特に断りのない限り**表 2** に示したとおりである．L1 キャッシュはコアごとに独立である一方，評価対象の L2 キャッシュは全コアで共有である．以降は単にキャッシュと表記した場合は共有 L2 キャッシュを

表 2　CPU とキャッシュの設定
Table 2　CPU and cache coufiguration.

各コアの構成	
コア数	4 または 2
CPU instruction set	Alpha
L1 cache	Instruction/Data 独立 16 KB/4-way/LRU 置換 line size 64 B/2-cycle latency
共有キャッシュ・メモリの構成	
Shared L2 cache	Instruction/Data 共有 1 MB/16-way/LRU 置換 line size 64 B/12-cycle latency 32 セットを計測セットとして使用
Memory	200-cycle latency

示す．本評価では提案した PTRP を他の動的キャッシュ分割方式と比較する．そのため同条件となるように本評価ではキャッシュの置換を LRU で行う．また UCP は DSS を適用して任意の 32 セットのみ計測対象とする．また CPAR についても UCP が計測を行う同一のセットを計測対象とする．さらに PTRP の計測セットも UCP で計測対象のセットと同一とする．

ベンチマークは SPEC CINT2000 から，キャッシュの way 数変化に対して異なるミス数変化をともなうプログラムから vpr，mcf，crafty，gap，vortex，twolf の 6 種類を使用する．入力データは ref を用いる．各プログラムは先頭 10 億命令実行後の状態から 10 億サイクルの実行で評価する．試行期間は特に断りがない限り 500 万サイクルとする．

4.2 ベンチマークプログラムのグループ分け

予備評価として SPEC CINT2000 の 6 種類のプログラムの単体実行時の，占有 way 数とキャッシュミス数の関係を**図 7** に示す．評価はキャッシュのセット数を一定に保ち，way 数を変化させた結果である．

評価結果から 6 種類のテストは主に 3 種類の特性を持つことが分かる．1 つは vpr，twolf のようにキャッシュの way 数に比例してミスが減少するという特性である．これらのプログラムをグループ A とする．もう 1 つは crafty，vortex のように way 数が少ないうちは way の増加にともない急激にミスが減少し，一定の way 数以上でミス率が 0 に近づきミスが減少しなくなるという特性である．これらのプログラムをグループ B とする．最後に mcf，gap のように way 数が少ない場合のみ way 数増加がミス率に影響し，ミスが一定量残る特

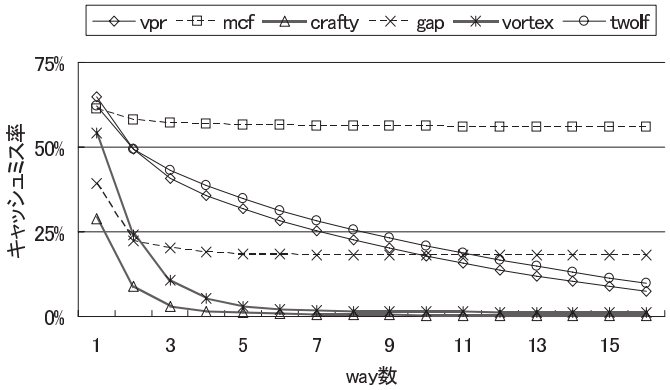

図 7　各プログラムの way 数と L2 キャッシュのミス率の関係
Fig. 7　Relation between cache ways assigned for each program and the L2 cache miss rate.

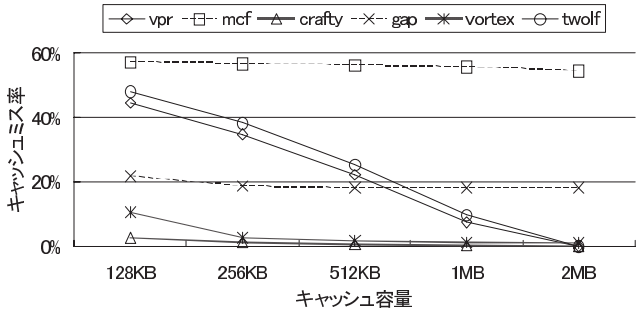

図 8　L2 キャッシュ容量と各プログラムのミス率の関係
Fig. 8　Relation between L2 cache capacity and the miss rate of each program.

性である．これらのプログラムをグループ C とする．以降の評価では 3 グループのプログラムの組合せ別に各方式のミス削減効果，性能向上効果を比較する．

また 6 種類のプログラムの特性が，容量と way 数のどちらに対する依存か確認するため，L2 キャッシュ容量に対する各プログラムのミス率の評価結果を**図 8** に示す．評価は way 数を 16 に固定し，セット数の変更のみでキャッシュ容量を調整する．図 8 と図 7 の結果を比較すると，グループ C は way 数，容量によらずミス率の変化が小さい．またグループ A において，図 7 における way 数が 2，4，8 の結果と，図 8 で対応する容量となる 128 KB，

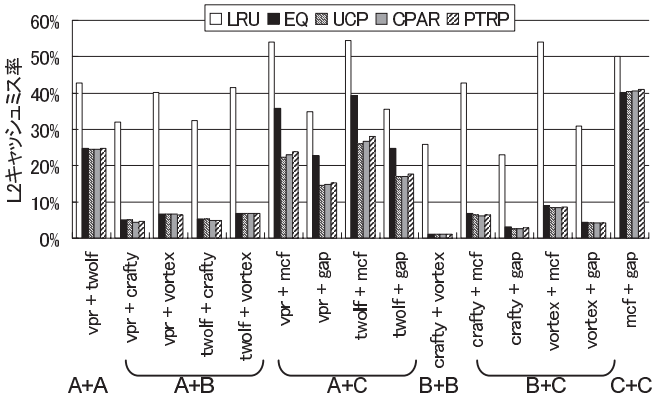

図 9　各キャッシュ分割方式適用時のキャッシュミス率（2 コア）
Fig. 9　Cache miss rate implementing each runtime cache partitioning method (2-core).

256 KB，512 KB の結果がほぼ等しい．つまりグループ A のミス率はキャッシュ容量に依存すると推定できる．一方グループ B では，キャッシュ容量が等しい 2 way と 128 KB，4 way と 256 KB での 2 つの結果のミス率を比較すると，way 数変更時のミス率が way 数固定時のミス率より高くなる．つまりグループ B のミス率は way 数に依存すると推測できる．

4.3　キャッシュミス削減量の評価・考察

まず 2 コアでのプログラム実行時の各方式における共有キャッシュのミス率の結果は**図 9** に示したとおりである．結果からキャッシュの分割によりキャッシュミス率が低下することが分かる．つまりキャッシュ分割を行わない場合に発生するミスの原因は，主に 2 プログラム間のキャッシュ上での競合が原因と推測できる．

次に実行プログラムのグループの組合せとキャッシュミス削減効果の関係を表す結果として，LRU 方式からのキャッシュミス削減比率を組合せ別に示す（**図 10**）．PTRP は LRU に比べて 18%から最大で 96%のキャッシュミスを削減している．また way の均等分割との比較で，PTRP は効果の高いグループ A，C の組合せで 31%のキャッシュミスを削減している．一方で PTRP の UCP，CPAR の適用時と比べたキャッシュミスの増加は平均で 0.4%，1.5%である．

PTRP を含めた 3 種類の動的キャッシュ分割方式によるミスの削減効果はグループ A と C の組合せで高い．一方他の組合せでは，均等分割と比べて大きなミス削減効果は得られていない．グループ A のプログラムは way 数に比例してミスが減少する一方，グループ C

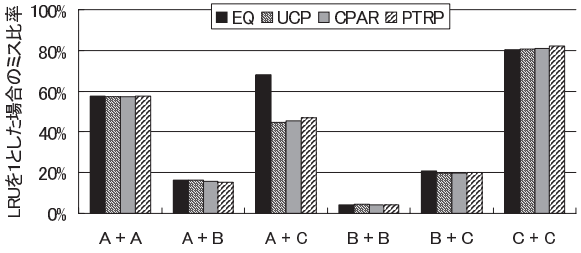

図 10　実行プログラムの組合せ別ミス削減効果（2 コア）
Fig. 10　Cache miss reduction effect of cache partitioning methods for each pair of programs (2-core).

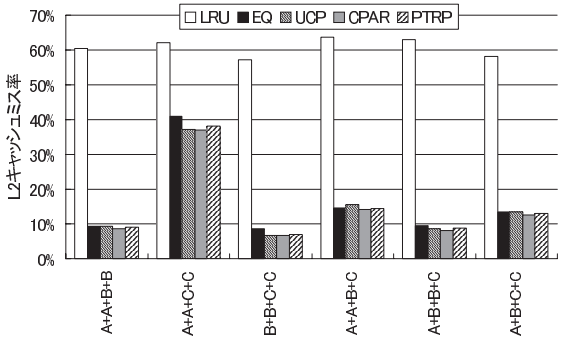

図 11　各キャッシュ分割方式適用時のキャッシュミス率（4 コア）
Fig. 11　Cache miss rate implementing each runtime cache partitioning method (4-core).

はミスの way 数への依存が低い．つまりグループ A と C の組合せではキャッシュ分割による競合ミス発生防止に加えて，動的な分割 way 数変更によるミス最小化の効果が現れていると推測できる．

続いて 4 コアでのプログラム実行時の各方式における共有キャッシュのミス率は**図 11** に示したとおりである．結果は実行プログラムのグループの組合せ別に示している．4 コアの場合も 2 コアと同様に，プログラムの種類によらずキャッシュの分割によりミスが削減されることが分かる．また PTRP を含めた 3 種類の動的キャッシュ分割方式によるミスの削減効果は，グループ C を 2 コアで実行した場合に最も大きいことが分かる．これは 2 コアの結果と同様に，キャッシュの way 数増加の効果が低いグループ C のプログラムの分割 way 数が削減されたことが理由と推測できる．

4 コアの場合 PTRP は LRU に比べて 38%から最大で 88%のキャッシュミスを削減している．また単純な均等分割との比較では最大で 19%のミスを削減している．さらに UCP と比べて，平均で 2.8%のキャッシュミスを削減している．一方 CPAR と比べた場合，キャッシュミスが平均で 3%増加する．よって PTRP は従来方式に近い性能と，従来方式では対応が困難な LRU 以外の置換方式への対応を両立させることが分かる．

また評価結果ではコア数によらず，すべての way のヒット数を計測する UCP より一部の way のヒット数のみを計測する CPAR でより高いミス削減効果が得られている．その理由は UCP の最適 way 分割の反映が 1 回計測で済む性質が原因と推測できる．UCP はキャッシュ利用状況の変化を 1 回の計測で way 分割に反映するため，一時的な処理等にともなうミス率計測のノイズの影響を受けやすい．つまり UCP は計測結果を直後の way 分割に反映するため，ノイズの影響でミスが増加すると推測できる．逆に CPAR，PTRP は各コアの way 数を 1 ずつ変化させるため，ミス率計測でのノイズ発生時に影響が小さいと推測できる．

4.4　IPC の評価・考察

PTRP によるキャッシュミス削減が実行速度に与える影響を他の方式と比較する．そのため各プログラムの，単体実行時の IPC に対する並列実行時の IPC の比率で各プログラムの処理性能を評価する．これらの合計値が各方式適用時の CPU 全体の処理性能を表す．2 コアの場合に各プログラムの IPC の，単体実行時の IPC 値との比率の合計値を求めた結果を**図 12** に示す．

CPU 全体の処理性能はキャッシュミス削減効果と同様にいずれの場合もキャッシュ分割により増加している．キャッシュミス削減効果の結果と比較すると，PTRP はミス削減効果が高いプログラムに対して特に高い性能向上効果を持つことが分かる．また PTRP はキャッシュ未分割時，均等分割時と比較して他の動的分割方式と同様の性能向上効果の傾向を示す．

また 4 コアの場合に各プログラムの IPC の，単体実行時の IPC 値との比率の合計値を求めた結果を**図 13** に示す．結果は実行プログラムのグループの組合せ別に示している．4 コアでも 2 コアと同様に，CPU 全体の IPC はキャッシュ分割により向上する．また PTRP を含めた動的キャッシュ分割は，キャッシュミスの削減効果が高いグループ B，C のプログラムを 2 コアずつ実行した場合に，特に均等分割と比べて高い性能向上効果を示すことが分かる．一方同様にキャッシュミスの削減効果が高かったグループ A，C のプログラムを 2 コアずつ実行した場合は，グループ B と C の場合と比べて動的分割の効果が低い．これは

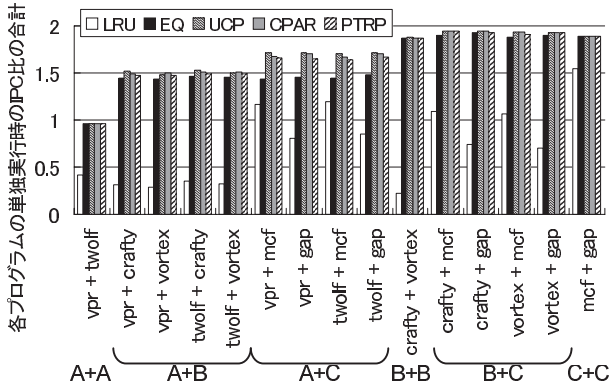

図 12　各プログラムの単独実行時からの性能向上比率（2 コア）
Fig. 12　IPC improvement ratio from single execution IPC of each program (2-core).

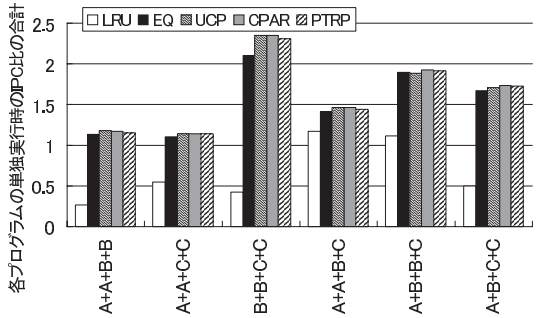

図 13　各プログラムの単独実行時からの性能向上比率（4 コア）
Fig. 13　IPC improvement ratio from single execution IPC of each program (4-core).

図 11 の結果から，グループ A と C の組合せでは動的キャッシュ分割適用時のキャッシュミス削減幅が他の組合せより小さいことが理由と推測できる．

4.5　試行によるミス増加量の評価・考察

PTRP において，キャッシュミス計測に計測セットを用いることによるミス削減効果を評価する．計測セットを用いない場合，最適 way 分割の探索にキャッシュ全体の way 分割を変更して way 数増減時のミス率を計測する必要がある．以下，PTRP に対して計測セットを用いない最適 way 分割方式を TRP（Trial-Based Runtime Partitioning）と定義する．

表 3　計測セットの使用によるミス削減効果（4 コア）
Table 3　Cache miss reduction effect of using “monitoring sets”.

プログラム	ミス率		PTRP 由来のミス割合
	TRP	PTRP（TRP 比）	
A+A+B+B	9.5%	9.1%（−4.0%）	0.23%
A+A+C+C	40%	38%（−3.6%）	0.04%
B+B+C+C	8.6%	7.0%（−4.4%）	0.12%
A+A+B+C	15%	14%（−2.8%）	0.10%
A+B+B+C	9.6%	9.0%（−3.0%）	0.17%
A+B+C+C	13%	13%（−2.9%）	0.08%

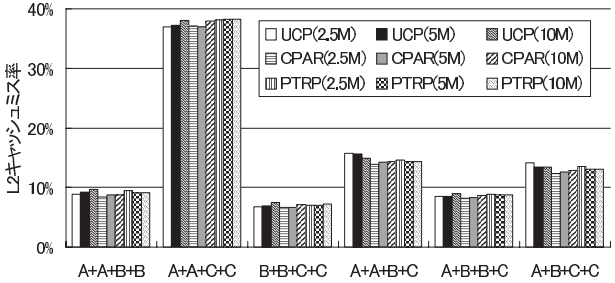

図 14　キャッシュミス削減効果と試行期間の長さの関係（4 コア）
Fig. 14　Relation between cache miss reduction effect of partitioning and trial period length (4-core).

4 コアでの PTRP および TRP のキャッシュミス率，および PTRP での部分的試行に由来するミスが全ミスに占める割合の結果を**表 3** に示す．TRP は，図 11 で示したキャッシュ未分割の場合，および均等分割の場合と比べてキャッシュミスを低減させている．さらに PTRP と TRP のキャッシュミス率と比較すると，PTRP は計測セットを用いることで TRP と比べて 2.8%から 4.4%のキャッシュミスを削減することが分かる．

また表 3 の結果から，PTRP での部分的な way 分割の試行で余分に発生したキャッシュミスの割合は全ミスの 0.04%から最大で 0.23%であることが分かる．つまり PTRP によるキャッシュミス増加量は，図 9，図 11 で示された PTRP のキャッシュミス削減効果と比べて十分に小さいことが分かる．

4.6　最適分割発見と試行期間の関係の評価・考察

試行期間の長さとミス削減効果の関係の評価結果を**図 14** に示す．評価は 4 コアで行い，

表 4　各機能の実現に必要なハードウェア資源（4 コア）
Table 4　The amount of hardware resource implementing each function (4-core).

	セットごと（32 セット）			コアごと	
	タグ	コア ID	LRU	カウンタ	way 数
	16 bit	2 bit	4 bit	32 bit	4 bit
UCP	16	16	16	16+1	1
CPAR	1	0	0	2+1	1
PTRP	0	0	0	2	2

試行期間が 2.5 M，5 M，10 M サイクルである場合の各プログラムの組合せに対する UCP，CPAR，PTRP のミス率を表す．PTRP の最適 way 分割の発見時間は，コア数に非依存である．一方で PTRP は試行期間ごとにキャッシュミス率計測，way 分割の変更を行うため，最適 way 分割の発見時間が試行期間の長さに比例する．つまり試行期間の長さが PTRP のミス削減効果に影響を与える．

図 14 の結果では，同一プログラムの組合せに対する PTRP のキャッシュミス削減効果について，ミス率が最大の試行期間では最小の場合に比べて平均で 2.6%ミスが多く発生する．また UCP，CPAR，PTRP すべてのキャッシュミス率について試行期間の長さとの関連性は見られない．つまり CPU 内で組合せを変更せずに実行する場合は最適な way 分割の探索を行う頻度が低いため，試行期間の長さが PTRP のミス削減効果に与える影響が小さいと推測できる．

4.7　追加ハードウェア資源量の評価・考察

最後に PTRP と UCP，CPAR について，実装に必要な追加ハードウェア量を記憶容量で比較する．まず各要素の資源追加量を表した結果は**表 4** に示したとおりである．評価では各方式で共通のキャッシュブロック本体，タグ，LRU の制御機能は省略する．また各方式で共通に用いる，各ブロックの占有コア情報の記憶領域も省略する．結果の各項目はキャッシュ外に追加が必要な要素を表す．タグはキャッシュヒットの判定，コア ID はブロックを占有するコアの判別，LRU は置換に使用する LRU の順序情報，カウンタはヒット数，アクセス数の計測，way 数はコアが占有可能な最大 way 数を表す目的で使用する．また CPU は 4 コアとして，その他の条件は性能評価と同一とする．

表 4 から UCP は他の 2 方式と比べて多くの資源を要することが分かる．これは各コアに対して way ごとにヒット数の計測機能を実装することによる．また UCP では表 4 以外に LRU の置換順序を制御する機能が別途必要である．CPAR は LRU の前後 1 way のヒット数に加えて全体のアクセス数の取得が必要であるためカウンタが 3 個必要である．PTRP は計測セットのためにアクセス数とヒット数のカウンタ，および余分に占有可能 way 数の情報が必要である．

表 5　各方式で必要な CPU 全体のハードウェア資源量（4 コア）
Table 5　The amount of hardware resource implementing each partitioning method (4-core).

	総容量	容量比率（PTRP=1）
UCP	5,906 B	160
CPAR	306 B	8.5
PTRP	36 B	1

次に本評価環境に対する CPU 全体の必要資源量を**表 5** に示す．比較の結果 PTRP は UCP と比べて追加資源が約 0.6%にとどまる．また CPAR との比較では追加資源の容量が約 12%であることが分かる．本評価は 4 コア，16-way の CPU で行ったが，さらにコア数，way 数が多いキャッシュでは必要資源量の差がさらに開くと推測できる．以上の評価から PTRP は従来方式と比べて LRU 以外の置換方式に対応しつつ，ハードウェアの記憶容量を約 12%に削減することが分かる．

5. ま と め

本論文では共有キャッシュで発生するミスを削減する部分的試行に基づく動的共有キャッシュ分割方式（PTRP）を提案した．PTRP は LRU に加えて任意の置換方式のキャッシュへの対応を特徴とする．また任意の置換方式に対応するため，キャッシュの一部分の way 分割を実際に変更することで，計測の性能への影響を低減させつつミスを計測する部分的試行と，部分的試行を行うためのキャッシュ構成を提案した．さらに部分的試行を用いた，キャッシュミスを最小化する way 分割の探索手順を提案した．評価により PTRP は均等分割と比べて 2 コアで最大 31%，4 コアで最大 19%のキャッシュミス削減効果を確認した．また従来のキャッシュ分割方式と比べて 2 コアで平均約 1.5%，4 コアで平均約 3%のキャッシュミス増加で済む一方で，追加資源の容量は従来方式の約 12%であることを確認した．以上の結果から本論文では PTRP による動的キャッシュ分割の有用性を示した．

本研究の今後の課題として，占有 way 数に対して non-convex なミス数変化をするプログラムへの対応，最適 way 分割探索の高精度化があげられる．前者は最適 way 分割を探索する際に non-convex の特性である，way 数変化に対する急激なミス数増減の検出が課題である．後者は特に疑似 LRU による置換方式に対応した，way 数増減によるミス数変化のハードウェアまたはソフトウェア的な推定方式の確立が課題である．

参 考 文 献

1) Albonesi, D.: Selective cache ways: On-demand cache resource allocation, *Proc. 32nd Annual International Symposium on Microarchitecture*, pp.248–259 (1999).
2) Yang, S., Powell, M., Falsafi, B., Roy, K. and Vijaykumar, T.: An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches, *Proc. 7th International Symposium on High-Performance Computer Architecture*, pp.147–158 (2001).
3) Yang, S., Powell, M., Falsafi, B. and Vijaykumar, T.: Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay, *Proc. 8th International Symposium on High-Performance Computer Architecture*, pp.151–161 (2002).
4) Chang, J. and Sohi, G.: Cooperative Caching for Chip Multiprocessors, *Proc. 33rd Annual International Symposium on Computer Architecture*, pp.264–276 (2006).
5) Chang, J. and Sohi, G.: Cooperative Cache Partitioning for Chip Multiprocessors, *Proc. 21st ACM International Conference on Supercomputing*, pp.242–252 (2007).
6) Kim, S., Chandra, D. and Solihin, Y.: Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture, *Proc. 13th International Conference on Parallel Architecture and Compilation Techniques*, pp.111–122 (2004).
7) Iyer, R.: CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms, *Proc. 18th ACM International Conference on Supercomputing*, pp.257–266 (2004).
8) Suh, G., Devadas, S. and Rudolph, L.: A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning, *Proc. 8th International Symposium on High-Performance Computer Architecture*, pp.117–128 (2002).
9) Suh, G., Rudolph, L. and Devadas, S.: Dynamic Partitioning of Shared Cache Memory, *Journal of Supercomputing*, Vol.28, No.1, pp.7–26 (2004).
10) Qureshi, M. and Patt, Y.: Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches, *Proc. 39th Annual International Symposium on Microarchitecture*, pp.423–432 (2006).
11) Stone, H., Turek, J. and Wolf, J.: Optimal Partitioning of Cache Memory, *IEEE Trans. Computers*, Vol.41, No.9, pp.1054–1068 (1992).
12) Qureshi, M., Lynch, D., Mutlu, O. and Patt, Y.: A case for MLP-aware cache replacement, *Proc. 33rd Annual International Symposium on Computer Architecture*, pp.167–178 (2006).
13) Dybdahl, H., Stenström, P. and Natvig, L.: A cache-partitioning aware replacement policy for chip multiprocessors, *Proc. 2006 ACM Conference on High Performance Computing*, pp.22–34 (2006).
14) Mattson, R., Gecsei, J., Slutz, D. and Traiger, I.: Evaluation techniques for storage hierarchies, *IBM Systems Journal*, Vol.9, No.2, pp.78–117 (1970).

(平成 21 年 1 月 27 日受付)
(平成 21 年 5 月 1 日採録)

小川　周吾（正会員）

2005 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了．現在，日本電気株式会社システムプラットフォーム研究所勤務．主にストレージシステムの研究に従事．2007 年より東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程在学中．電子情報通信学会員．

入江　英嗣（正会員）

1999 年東京大学工学部電子情報工学科卒業．2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了．博士（情報理工学）．2004 年より科学技術振興機構 CREST「ディペンダブル情報処理基盤」研究員．2008 年より東京大学情報理工学系研究科助教．コンピュータ・システム，特にプロセッサアーキテクチャの研究に従事．電子情報通信学会コンピュータシステム専門研究委員会幹事補佐（2007 年〜）．電子情報通信学会，ACM 各会員．

平木　　敬（正会員）

東京大学理学部物理学科，東京大学理学系研究科物理学専門課程博士課程退学，理学博士．工業技術院電子技術総合研究所，米国 IBM 社 T.J. Watson 研究センターを経て現在東京大学大学院情報理工学系研究科勤務．数式処理計算機 FLATS，データフロースーパコンピュータ SIGMA-1，大規模共有メモリ計算機 JUMP-1 等，多くのコンピュータシステムの研究開発に従事，現在は超高速ネットワークを用いる遠隔データ共有システム Data Reservoir システムの研究，超高速計算システム GRAPE-DR の研究を行っている．