Vol.2009-ARC-184 No.20
2009/8/5

置換データの性質に着目した動的キャッシュパーティショニング

小川周吾[†1]　入江英嗣[†1]　平木敬[†1]

マルチコア CPU の共有キャッシュでは競合によるミスが発生する．その回避手段として，動的キャッシュパーティショニング方式が提案されている．しかし従来のキャッシュパーティショニング方式では，必要なハードウェア量，またはソフトウェアの制御に伴う複雑性が問題である．そこで本論文では victim キャッシュを用いた，共有キャッシュのミスを最小化する動的パーティショニング方式である Victim-Guided Cache Partitioning (VGCP) を提案する．VGCP では共有キャッシュに対してコア毎に独立した victim キャッシュを実装し，そのヒット数から共有キャッシュのパーティショニングを決定する．VGCP に関する性能評価の結果，多くのプログラムの実行時は従来の動的なパーティショニング方式と比べて 2 コアの場合で最大 4%のミス増加にとどまることを確認した．しかし VGCP は特定のプログラム実行時に従来方式と比べてキャッシュミスが最大 41% 増加する問題点を併せて確認した．

Runtime Cache Partitioning Using Victim Behavior

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

To avoid the increase of conflict misses among cores which share caches on multicore CPU, many dynamic cache partitioning method have been proposed. However these methods have complexity that originates in the amount of hardware resource, or the necessity of software management. In this paper, we propose Victim-Guided Cache Partitioning (VGCP) for minimizing cache misses in shared caches using victim cache information. VGCP implements victim caches for each core, then VGCP partitions shared caches for minimizing shared cache misses using hit frequency information of each victim caches. We evaluate an effect for performance of VGCP, then we show that VGCP reduces hardware and software complexity with 4% cache miss increase in dual-core CPU at the maximum. However we also show that VGCP increases cache misses with 41% cache miss increase at the maximum.

1. はじめに

近年普及しているマルチコア CPU では複数コアが異なるプロセスを並列に実行可能である．マルチコア CPU ではプロセスの並列実行数の増加に伴い，CPU 全体で単一コアの場合より多くのメモリアクセスが発生する．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．またキャッシュの共有により，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．更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．

そこでコア間の競合ミス発生を避けるために共有キャッシュの動的パーティショニング方式が提案されている．キャッシュパーティショニングでは共有キャッシュを各コアが占有可能な領域に分割することで競合を防止する．この動的パーティショニングを用いることで，共有キャッシュ全体で発生するミス数を最小化する方式が提案されている[2)3)4)]．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から各コアの新たなキャッシュ占有量を決定する．

しかしこれらの動的キャッシュパーティショニング方式は，ミス数の計測に多くのハードウェア資源が必要である．そのため実装の複雑性が課題である．また LRU の性質を用いてミス数を計測するため，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．

本論文では実装が容易であり任意の置換方式のキャッシュに適用可能である，Victim-Guided Cache Partitioning（VGCP）を提案する．VGP では共有キャッシュとメモリの間に victim キャッシュを用意する．この victim キャッシュへのヒットの頻度から共有キャッシュのパーティショニングを決定する．

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

Vol.2009-ARC-184 No.20
2009/8/5

2. 関連研究

2.1 ミスを最小化する動的共有キャッシュパーティショニング

Stone ら[1] は複数のプロセスが利用するキャッシュについて，ミスを最小化する way 単位のパーティショニング方式を提案した．Stone らの方式では各プロセスが使用するキャッシュの way 数とミス数の関係を事前に調べる．この結果からキャッシュ全体のミスが最小となる way のパーティショニングを求める greedy アルゴリズムを提案した．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り当て way 数を変化させた場合のミス回数の記録が必要である．

Suh ら[2][3] 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，共有キャッシュを動的に way 単位でパーティショニングする方式を提案した．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，gain が最大値となる way 単位のパーティショニングを探索することでミスを低減する．しかし Suh らの提案方式は，CPU 上の各コアに対して way 毎の 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[4] は，キャッシュの外部で marginal gain（Qureshi らは”marginal utility” と定義）を計測して動的キャッシュパーティショニングを行う方式として Utility-Based Cache Partitioning（以下 UCP）を提案した．UCP ではコア毎に gain を記録するカウンタ，キャッシュヒットの判定に用いるタグ情報を利用する．また Suh らのアルゴリズムより少ない計算量でキャッシュミスを最小化するアルゴリズムを提案した．更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）[5] を用いることで，Suh らの方式と比べてハードウェア量を削減した．

また Dybdahl ら[7] は，各コアが占有可能な共有キャッシュの way 数を 1 増減した場合のミス数を予測して，キャッシュミスを低減するようにパーティショニングを変更する Cache-Partitioning Aware Replacement Policy（以下 CPAR）を提案した．この方式はキャッシュの各セットにコア毎に，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．

一方我々は way 毎のミス数を計測する方式とは異なる動的キャッシュパーティショニング方式として Partial Trial-Based Runtime Partitioning[9]（以下 PTRP）を提案した．PTRP はキャッシュの少量のセットを用いて部分的にパーティショニングの変更を試行することで，

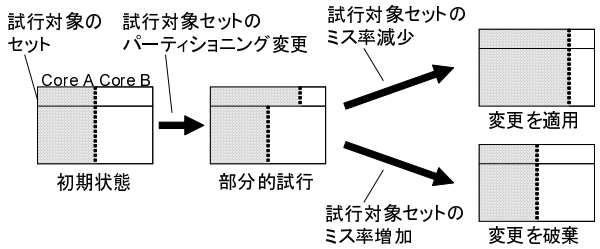

図 1 部分的試行を利用したパーティショニング決定

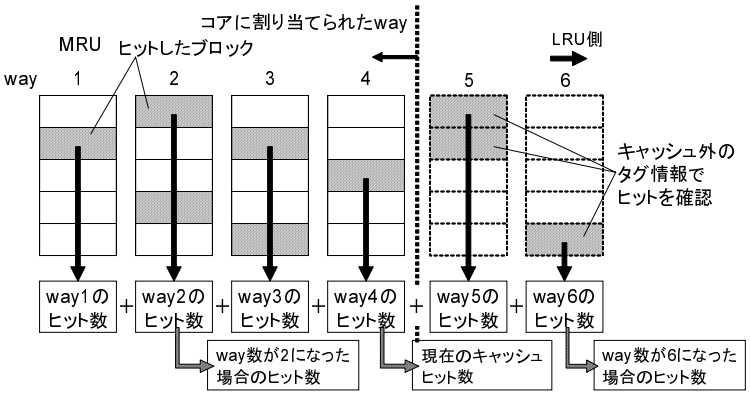

図 2 キャッシュの MRU から LRU までの way とヒット数のカウンタ

ミスを最小化するパーティショニングを探索する（図 1）．これによりキャッシュ置換方式への依存の回避と，必要なハードウェア量の削減を実現した．

2.2 従来のキャッシュパーティショニングの課題

多くの動的キャッシュパーティショニング方式[3][4][7] では，各コアについて占有する way 数毎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．従来方式のうち UCP では取り得る全ての way 数に対して，CPAR では現在の占有 way 数に対して 1 増減した場合のミス数のみを計測する．

取り得る各 way 数でのキャッシュミス数を同時に計測するには図 2 に示した通り，コア毎に共有キャッシュの way 数と同数のカウンタを準備する．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．例えばヒットしたブロックの

Vol.2009-ARC-184 No.20
2009/8/5

MRU から数えた順番が n の場合，n 番目のヒット回数を記録するカウンタの値を更新する．カウンタに記録されたヒット回数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．このときキャッシュミス数の算出に，LRU による置換の性質である stack property[6] を利用する．この性質から n-way を占有する場合のキャッシュヒット数は MRU から n-way までに対応するカウンタに記録されたヒット数の合計で求められる．

よって stack property を用いてミス数を計測するキャッシュパーティショニング方式は，多くのハードウェアを必要とする点が第 1 の課題である．stack property を用いたパーティショニング方式では，実際のキャッシュ上に存在しないデータに対するヒットの有無を判定するために，共有キャッシュ上の計測を行う各セットに対してコア毎に全 way 数分のタグ情報が必要である．そのため必要なハードウェア量は CPU コア数，共有キャッシュの way 数に比例する．これらの資源をキャッシュ外に持つため，実装が複雑である．

第 2 の課題は 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である．例えば LRU の代替置換方式である疑似 LRU は，LRU が持つ各セット内のブロックがアクセスされた時間順序の情報を簡略化することで実装を容易にした方式である．疑似 LRU では時間順序情報が簡略化されるため，LRU とはブロックの置換順序が異なる．つまり stack property は成立せず，各占有 way 数に対するミス数の計測は困難である．

一方 PTRP は stack property を利用しないため，キャッシュ上に存在しないデータのヒット判定は不要である．そのため他のキャッシュパーティショニング方式と比べて実装に必要なハードウェア資源量が少ない．また LRU 以外の置換方式を用いたキャッシュへの適用が可能である．しかし PTRP では最適なパーティショニングの探索を部分的な試行を繰り返すことで実現する．そのため最適パーティショニングの探索に他の方式より多くの手順を要する課題を持つ．また最適パーティショニングの探索にソフトウェアの補助が多く必要であるため，ソフトウェアオーバーヘッドの発生が課題である．

3. Victim-Guided Cache Partitioning（VGCP）

本論文ではマルチコア CPU の共有キャッシュパーティショニング方式として，Victim-Guided Cache Partitioning（VGCP）を提案する．VGCP では共有キャッシュに対する victim キャッシュを実装し，victim キャッシュのアクセス数，ヒット数からパーティショニングを決定する．

3.1 victim キャッシュを用いたミス数削減効果の予測方式

マルチコア CPU の共有キャッシュで発生するミスを最小化するパーティショニングを求めるためには，各コアへの共有キャッシュの way 割り当て増減に対するミス数変化の予測が必要がある．しかし従来のパーティショニング方式の多くは way の増減によるミス数の変化を予測するために多くのハードウェア資源，またはソフトウェアによる定期的な制御を必要とする．よって従来のパーティショニング方式は何れも実装の複雑性が課題である．そのため共有キャッシュの新たなパーティショニング方式では，各コアへの way 割り当て増減時のミス変化量予測を少ないハードウェア資源と少ない制御で行うことが必要である．

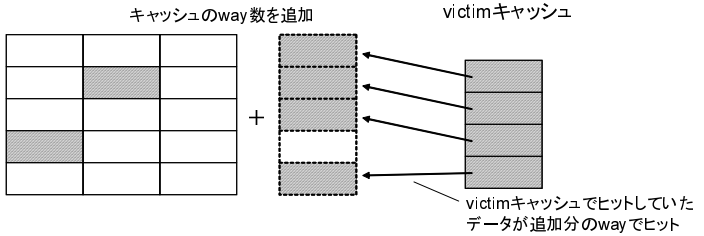

図 3 キャッシュの way 数と victim キャッシュのヒット数の関係

本論文では各コアに対する way の割り当て増減によるミス数変化を調べるために，共有キャッシュに対して victim キャッシュ[8] を導入する．victim キャッシュは一般的にキャッシュとメモリの間に実装され，上位のキャッシュから競合により追い出されたデータを格納する小容量のキャッシュである．victim キャッシュに追い出されたデータを格納することで，データへの再アクセス時にメモリへのアクセス発生を防ぐ効果を持つ．

victim キャッシュは一般に上位のキャッシュから追い出されたデータを LRU または疑似 LRU の置換方式に従って格納する．つまり小容量の victim キャッシュに格納されるデータは，上位のキャッシュから追い出された後に短時間でアクセスされたことを表す．上位のキャッシュから追い出されたデータについて，再アクセスまでの時間が短いほど上位キャッシュの同一セットに格納されるデータへのアクセス頻度が減少し，同一セットの競合発生頻度が減少する．そのため再アクセスまでの時間が短時間なデータほど，上位のキャッシュの way 数を増加することでキャッシュから追い出されずにとどまりやすくなる．

よって小容量の victim キャッシュに対して，ヒット数が多いほど共有キャッシュの way 数増加によるミス削減量が多いと推測できる（図 3）．しかし victim キャッシュが一般に full-associative であるのに対して共有キャッシュの増加分の way は set-associative であり構成が異なる．そのため 2 種類のキャッシュは，キャッシュへのアクセス発生頻度に対して格

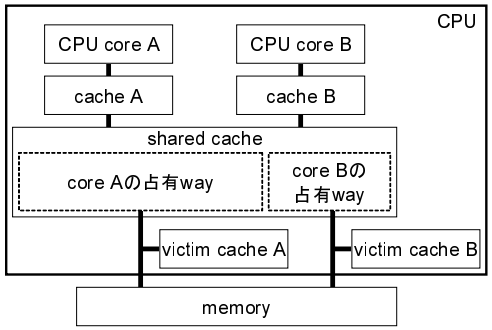

図 4 VGCP におけるマルチコア CPU のキャッシュ構成（2 コアの場合）

納するデータが追い出されるまでの時間の依存関係が異なる．つまり共有キャッシュのヒット数はキャッシュへのアクセス発生頻度にも依存する．そこで VGCP では各コアの victim キャッシュに対するヒット数とアクセス数から共有キャッシュの way 数の割り当て数増加時のミス削減効果量の評価値を求める．パーティショニング変更時は victim キャッシュの評価値が多いコアへの way 数の割り当てを増やす．

VGCP は victim キャッシュのヒット数を用いることで，従来のパーティショニング方式に比べてハードウェア資源の有効利用が可能である利点を持つ．従来の多くの方式では，各コアに対する共有キャッシュの way 割り当て増減に伴うミス数変化の予測のために専用のハードウェア資源を必要とする．一方 VGCP は共有キャッシュに対して新たに victim キャッシュのみを必要とする．victim キャッシュはミス数変化の予測機能だけではなく，メモリアクセスを削減するキャッシュとしてハードウェア資源を有効活用できる．また VGCP は victim キャッシュのアクセス数，ヒット数のみでパーティショニングを決定するため従来方式に比べて実装がより単純である利点を持つ．

3.2 VGCP の実装

3.2.1 CPU 全体のキャッシュ構成

VGCP を適用するマルチコア CPU のキャッシュ構成は図 4 に示した通りである．図 4 の CPU はコア毎のキャッシュとコア間の共有キャッシュを持つ．共有キャッシュのパーティショニングは従来方式[3)4)] と同様に，キャッシュ内の各セットにおける各コアの占有 way 数の制限により実現する．また VGCP によるパーティショニング決定のため victim キャッシュが必要である．victim キャッシュは共有キャッシュとメモリとの間に，コア毎に同容量存在する．victim キャッシュの容量は各コアに割り当てられた共有キャッシュより小容量である．

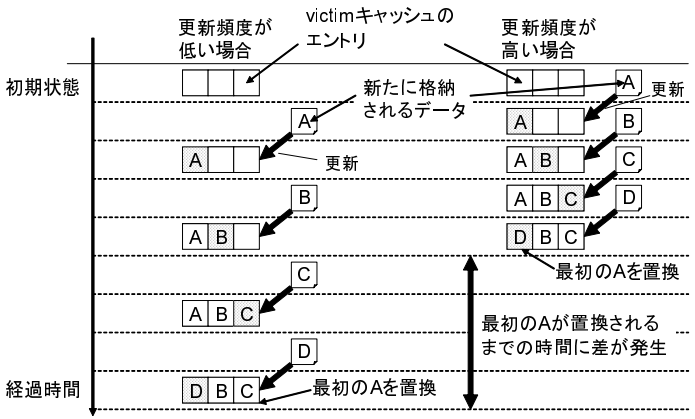

図 5 victim キャッシュからデータが置換されるまでの時間とデータ更新頻度の関係

VGCP ではコア毎に victim キャッシュを分けることで，他のコアの影響を受けずに victim キャッシュのアクセス数，ヒット数を計測する．これらの計測を一定時間毎に行い，パーティショニング決定の指標に利用する．パーティショニング変更は victim キャッシュのヒット数から，割り当てる way を増やした場合のミス数削減効果が最大と予測されるコアの way 数を増やすことで行う．パーティショニングの変更を繰り返すことで，共有キャッシュ全体のミス数が最小となるパーティショニングを探索する．

3.2.2 パーティショニングを決定する評価値の算出

VGCP では各コアに対応した victim キャッシュのアクセス回数，ヒット回数を計測する．計測した各コアのヒット回数から共有キャッシュのパーティショニングを決定する．しかし共有キャッシュと victim キャッシュは互いに構成，性質が異なる．

そのため各コアの victim キャッシュにおけるヒット数は，各コアへの共有キャッシュの割り当て way 数増加時のミス減少量を必ずしも直接的に示さない．共有キャッシュと victim キャッシュが異なる点は図 5 に示した通り，victim キャッシュでは victim キャッシュそのものに対するアクセス頻度がキャッシュ内のデータを保持する時間に影響する点である．victim キャッシュは新たなデータを格納する際に古いデータを優先的に置換する．この性質により一般的な full-associative の victim キャッシュでは，キャッシュ内のデータの更新毎に既存の格納データが置換により追い出される．

よってデータが victim キャッシュに格納後，同じコアの victim キャッシュでエントリ数に等しい数のミスが発生するとデータは必ず置換により victim キャッシュから追い出される．つまり victim キャッシュのデータは，データの格納後エントリ数に等しい回数のミスが発生する前にのみヒットする．そのため victim キャッシュ上の各データが追い出される前にヒットする確率は，victim キャッシュに対するデータのアクセス，更新頻度に依存しない．

一方共有キャッシュは一般的に set-associative であるため，キャッシュ上のデータの更新は，更新対象データの格納先のセット以外には影響を及ぼさない．よって共有キャッシュのセット数が十分に多く，各セットに対して偏りなくアクセスが行われる条件であれば，共有キャッシュで競合による特定のデータの追い出しが発生する確率は victim キャッシュに比べて無視できる．そのため共有キャッシュ内の各データが追い出される前にヒットする確率は，共有キャッシュに対するアクセス頻度に比例する．

つまり victim キャッシュのヒット数を共有キャッシュの割り当て way 数を増やすコアを決定するためのための評価値に用いる場合，共有キャッシュと victim キャッシュの各データがヒットする確率の違いを考慮して補正する必要がある．具体的には victim キャッシュのヒット数を，共有キャッシュの性質である各データのヒット率が各コアの victim キャッシュへのアクセス数の比率で補正した値が，割り当て way 数を増やすコアを決定するための評価値となる．コア i における victim キャッシュへの一定時間でのアクセス回数を n_i，ヒット回数を h_i とした場合の，VGCP におけるパーティショニング決定の評価値である補正後のヒット数 H_i は以下の通り求められる．

$$H_i = h_i \times n_i \quad (1)$$

3.2.3 評価値を用いた最適パーティショニングの決定

VGCP では一定時間毎に victim キャッシュに対するアクセス数，ヒット数を計測する．そして victim キャッシュのヒット数から求められた評価値をコア間で比較する．VGCP は評価値の結果から共有キャッシュのパーティショニングを決定し，再びアクセス数，ヒット数の計測を開始する．パーティショニングの変更を繰り返すことで，キャッシュミスを最小化するパーティショニングを探索する．

VGCP において victim キャッシュに対する各計測後の共有キャッシュのパーティショニング変更では，way の割り当てを 1 増やすコア，1 減らすコアを 1 つずつ決定する．まず割り当てる way 数を増やすコアの決定では前述の，victim キャッシュのアクセス数，ヒット数から求められるパーティショニング決定の評価値 H_i を用いる．victim キャッシュのヒット数が多く評価値が最大のコアに対して，共有キャッシュの割り当て way 数を増やすことにより最大のミス削減効果が得られると推測できる．

表 1 CPU とキャッシュの設定

各コアの構成	
コア数	4 または 2
CPU instruction set	Alpha
L1 cache	Instruction/Data 独立/ 16KB/ 4-way/ LRU 置換 line size 64B/ 2-cycle latency
共有キャッシュの構成	
Shared L2 cache	Instruction/Data 共有/ 1MB/ 16-way/ LRU 置換 line size 64B/12-cycle latency UCP/CPAR/PTRP は 32 セットのみ計測対象
victim キャッシュ・メモリの構成	
Victim cache	Instruction/Data 共有/ 32 entries, full-associative/ LRU 置換 line size 64B/ 10-cycle latency
Memory	200-cycle latency

一方 CPU 内で共有キャッシュの割り当て way 数を増やすコアが存在する場合，他のコアへの割り当て way 数を減らす必要がある．CPU 全体のキャッシュミス数を最小化するためには，割り当てる way 数を減らした場合のミス増加幅が最小のコアを選択する必要がある．本論文では共有キャッシュの割り当て way 数を増やすコアを決定するための評価値が最低であったコアについて，割り当てる way 数を減らした場合のミス増加量が最小であると仮定する．つまり VGCP では評価値が最大のコアに対して割り当てる way 数を 1 増やし，最小のコアに対して割り当てる way 数を 1 減らすことで，パーティショニングを変更する．

4. 性能評価

提案した VGCP と従来の共有キャッシュ全体のミスを最小化する動的キャッシュパーティショニング方式の性能を評価する．

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)，コア毎に 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，PTRP 及び本研究で提案する VGCP で動的にパーティショニングを行った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を用いる．

シミュレーションパラメータを述べる．CPU 及びキャッシュは特に断りのない限り表 1 に示した通りである．L1 キャッシュはコア毎に独立である一方，評価対象の L2 キャッシュは全コアが共有する．以降は単にキャッシュと表記した場合は共有 L2 キャッシュを示す．本

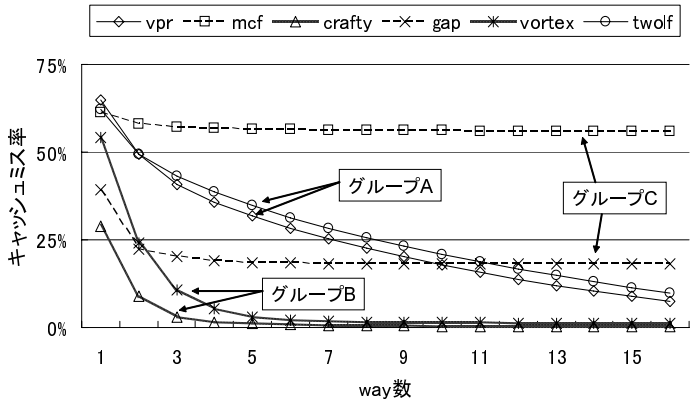

図 6 各プログラムの way 数と L2 キャッシュのミス率の関係

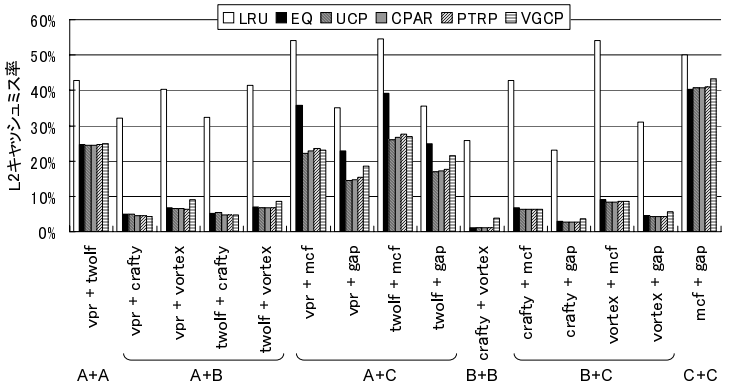

図 7 各キャッシュパーティショニング方式適用時のミス率 (2 コア)

評価では提案した VGCP と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．加えて 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．更に PTRP で部分的なパーティショニング試行を行うためのセットも UCP で計測対象のセットと同一とする．

victim キャッシュは本研究で提案した VGCP のみで使用する．また victim キャッシュはコア毎に互いに独立である．victim キャッシュを使用する VGCP では共有 L2 キャッシュでミスが発生すると victim キャッシュへのアクセスが発生し，victim キャッシュでもミスが発生した場合のみメモリへのアクセスを行う．

ベンチマークは我々が PTRP に対して行った評価[9] と同様に，SPEC CINT2000 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．アクセス数，ヒット数の計測，及びパーティショニングの変更間隔は 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

性能評価の前に 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 6 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 の増加に伴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み合わせを考慮して各方式のミス削減効果，性能向上効果を比較する．

4.3 2 コアでのキャッシュミス削減量の評価・考察

まず 2 コアでの各方式における共有キャッシュのミス率の結果を図 7 に示す．また実行プログラムのグループの組み合わ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み合わせ別に示した結果を図 8 に併せて示す．図 7，図 8 のキャッシュミス率は，VGCP における victim キャッシュでのヒット分は含まない．

まずキャッシュパーティショニング方式に関わらず，コア毎にキャッシュを分けることで全てのプログラムの組み合わせでミス率が低下することが分かる．つまりキャッシュパーティショニングを行わない場合に発生するミスの原因は，主に複数コア間の共有キャッシュ上での競合であると推測できる．また共有キャッシュを均等に分割する EQ と動的パーティショニングを用いた場合を比較すると，動的パーティショニングが EQ に比べて高いミス削減効果を持つプログラムの組み合わせは図 8 よりグループ A，C の組み合わせのみである．

本論文で提案を行った VGCP と他のパーティショニング方式を比較すると，図 7，図 8

Vol.2009-ARC-184 No.20
2009/8/5

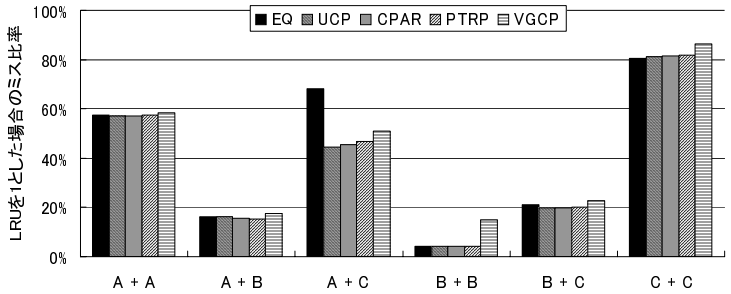

図 8 各パーティショニング方式の実行プログラムの組み合わせ別ミス削減効果 (2 コア)

で示した通り他の動的パーティショニングが EQ に比べて高いミス削減効果を示すプログラム A と C の組み合わせについて，同様に EQ より高いミス削減効果を示す．つまり VGCP は従来方式と同様に，共有キャッシュの動的パーティショニングによるミス削減効果を持つことを示す．しかし VGCP の効果は従来の動的パーティショニング方式に比べて低く，プログラム A と C の組み合わせでは UCP に比べて平均 14%のミス増加，よりミス削減効果の低い PTRP と比べて平均 9%のミス増加である．

続いて各プログラムの組み合わせに着目すると，図 7 より vortex，gap 以外の各プログラムを組み合わせて同時実行した場合に，VGCP は従来の動的パーティショニング方式に近いキャッシュミス削減効果を持つ．この場合 VGCP は従来方式と比べて最大 3%のミス削減を達成する．また VGCP の効果が最も低い場合でも，従来方式と比べて最大 4%のミス増加である．

しかし vortex, gap を組み合わせて実行した場合は VGCP のキャッシュミス削減効果は従来方式と比べて低下する．例えば crafty と vortex の組み合わせでは従来方式に比べてミスが 3 倍以上に増加する．この場合を除いても vortex 実行時は従来方式から最大で 41%ミスが増加している．加えて gap を含む組み合わせでも最大で 34%ミスが増加している．

これらの 2 種類のプログラム実行時に VGCP の効果が低い原因は，パーティショニング変更時の，way の割り当てを増やすべき，減らすべきコアを決定するための評価値の求め方が原因と推測できる．例えば vortex，gap は他のプログラムに比べて victim キャッシュのヒット数が多く，他のプログラムとアクセス特性が異なる．このアクセス特性の違いがパーティショニング決定の評価値に対して影響を与えると推測できる．また VGCP では割

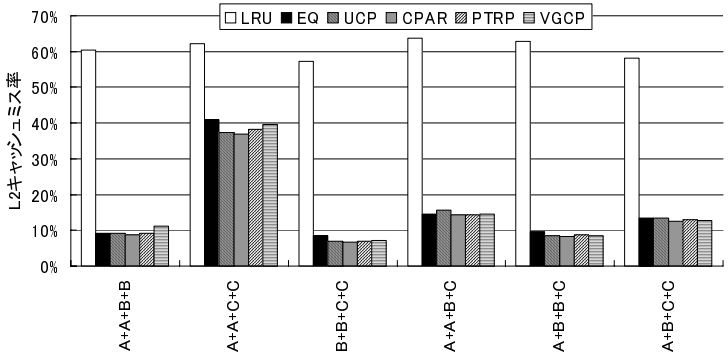

図 9 各キャッシュパーティショニング方式適用時のミス率 (4 コア)

り当てる way を増やすコアを決定するための評価値を，割り当てる way を減らすコアの決定に用いる．そのため way の割り当てを減らすコアの選択は必ずしも最適ではない．加えて victim キャッシュのヒット率が低い点も VGCP の効果が低い原因と推測できる．本論文の評価において victim キャッシュのヒット率は，共有キャッシュにおけるミス数の 0.1%以内である．そのため victim キャッシュのヒット数と，ヒット数から求められる評価値がプログラムの不規則かつ一時的な動作に伴う誤差の影響を受けやすくなる．

4.4 4 コアでのキャッシュミス削減量の評価・考察

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 9 に示した通りである．図 9 の結果は実行プログラムのグループの組み合わせ別に各パーティショニング方式適用時のミス率を表す．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．また VGCP を含む動的なパーティショニング方式によるミス削減効果は，2 コアでの評価結果と比べて EQ との差が小さい．

本論文で提案した VGCP に着目すると 2 コアでの評価結果とは異なり，グループ A，B から 2 種類ずつを同時に実行した場合を除き，全てのプログラムの組み合わせでキャッシュミス削減効果の大きな低下は見られず，従来方式に近い効果が得られる．従来方式と比べて VGCP 適用時のミス増加量は 1%から最大で 7%である．一方 VGCP によるキャッシュミス削減効果が低い，グループ A，B からの 2 種類ずつの同時実行では従来方式と比べて最大で 28%ミスが増加する．同一プログラムを実行しているにも関わらず 2 コアの場合に比べて vortex, gap によるキャッシュミス削減効果への影響が小さい原因として，まず同時実

行プログラム数が増加したため，各プログラムが CPU 全体に及ぼす影響が 2 コアの場合に比べて小さいことが理由であると推測できる．また同時実行プログラム数が増加したことで，共有キャッシュを 1 つのプログラムが大量に占有することが困難になることも理由であると推測できる．

5. ま と め

本論文では victim キャッシュを用いた動的な共有キャッシュパーティショニング方式である，Victim-Guided Cache Partitioning（VGCP）を提案した．VGCP は victim キャッシュを用いることで，パーティショニング専用のハードウェア，または複雑なソフトウェア制御を用いずにパーティショニングを行う．また VGCP を適用するマルチコア CPU の victim キャッシュの構成と，キャッシュパーティショニングの評価値として，victim キャッシュのヒット数をアクセス数で補正した値を利用する方式を提案した．評価により VGCP は動的キャッシュパーティショニングによるミス削減効果を持ち，従来方式と比べて多くのプログラムでは 2 コアで 4%，4 コアで 7%のミス増加で済むことを確認して有用性を示した．しかし VGCP 用いた場合，特定のプログラムを組み合わせた実行時に従来方式と比べて最大で 41%キャッシュミスが増加する問題点を併せて確認した．

本研究の今後の課題として，VGCP が従来のパーティショニング方式と比べてキャッシュミス削減効果が低下する詳細な状況の特定とその原因の究明が挙げられる．また共有キャッシュ，または victim キャッシュの容量，構成の違いによる VGCP のキャッシュミス削減効果の評価が挙げられる．更に，共有キャッシュの way 数の割り当てを減らすコアを決定する新たな評価値の提案も今後の課題である．

参 考 文 献

1) H. S. Stone, J. Turek, J. L. Wolf: Optimal partitioning of cache memory, IEEE Transactions on Computers, pp.1054–1068, 1992
2) G. E. Suh, S. Devadas, L. Rudolph: A new memory monitoring scheme for memory-aware scheduling and partitioning, In Proceedings of the 8th International Symposium on High-Performance Computer Architecture, pp.117–128, 2002
3) G. E. Suh, L. Rudolph, S. Devadas: Dynamic Partitioning of Shared Cache Memory, Journal of Supercomputing, 28, pp.7–26, 2004
4) M. K. Qureshi, Y. N. Patt: Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches, In Proceedings of the 39th Annual International Symposium on Microarchitecture, pp.423–432, 2006
5) M. K. Qureshi, D. N. Lynch, O. Mutlu, Y. N. Patt: A case for MLP-aware cache replacement, In Proceedings of the 33rd Annual International Symposium on Computer Architecture, pp.167–178, 2006
6) R. L. Mattson, J. Gecsei, D. R. Slutz, I. L. Traiger: Evaluation techniques for storage hierarchies, IBM Systems Journal 9(2), pp.78–117, 1970
7) H. Dybdahl, P. Stenström, L. Natvig: A cache-partitioning aware replacement policy for chip multiprocessors, In Proceedings of 2006 ACM Conference on High Performance Computing, pp.22–34, 2006
8) Norman P. Jouppi: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, In Proceedings of the 17th Annual International Symposium on Computer Architecture, pp.364-373, 1990
9) 小川 周吾, 入江 英嗣, 平木 敬: 部分的試行に基づく動的共有キャッシュ分割方式, 先進的計算基盤システムシンポジウム SACSIS2009, pp.369-378, 2009