

不要キャッシュブロックのパーティショニングによる排除方式

小 川 周 吾^{†2} 平 木 敬^{†1}

マルチコアプロセッサの共有キャッシュにおけるコア間の競合ミスを削減する複数のパーティショニング方式が提案されている。しかし多くのパーティショニング方式は、dead block と呼ばれるキャッシュ上の再アクセスされないブロックによる利用率の低下を考慮していない。本論文では我々が提案したマルチコアプロセッサ向けの共有キャッシュパーティショニング方式である HFCA (History-Free Cache Allocation) を利用して、共有キャッシュから dead block を排除し、各コアに効率的にキャッシュを割り当てる方式を提案する。HFCA は共有キャッシュを各コアのパーティションと、アクセス頻度の低いブロックを含んだ共有パーティションに分けることで dead block を分離する。評価の結果、SPEC CPU2006 と共有キャッシュより大きなデータへのアクセスが並列実行される場合に、HFCA により IPC が平均 6.5% 向上し、パーティションとして確保するキャッシュの way 数が全体の 57% に削減され、HFCA はキャッシュミスを最小化するパーティションの割り当て、及び dead block の排除効果を同時に実現できることが判明した。

1. はじめに

近年普及する多くのマルチコアプロセッサは、複数コアが単一の set-associative キャッシュを共有する。共有キャッシュは大容量のキャッシュの実現に必要なハードウェア量を削減し、一貫性維持の制御が不要でありコア間のデータ共有が容易である利点を持つ。一方で

複数コアがキャッシュを共有するため発生する、コア間の競合ミスによる性能低下が問題である。

コア間の競合ミスを削減するために、従来よりキャッシュパーティショニング方式が複数提案されている。これらのパーティショニング方式は、共有キャッシュを各コアの占有領域であるパーティションに分割することで、コア間のキャッシュ競合の発生を防ぐ。またキャッシュミス数を最小化するように各コアのパーティションの容量を決定する。しかしこれらの方式は各コアに割り当てた各キャッシュブロックのアクセス頻度を考慮しない。そのため各コアのパーティションに dead block と呼ばれる、キャッシュから追い出されるまでの間に再アクセスされないブロックが含まれ、キャッシュが効率的に利用されない問題を持つ。

本論文ではマルチコアプロセッサの共有キャッシュにおいて我々が提案したキャッシュパーティショニング方式である HFCA¹⁾ (History-Free Cache Allocation) を用いて各コアに対する dead block の割り当てを回避する方式を提案する。HFCA は共有キャッシュを各コアがアクセスするデータを格納する占有パーティションと、dead block を含んだ共有パーティションに分離する。HFCA により少量のハードウェアでキャッシュパーティショニングを行い、dead block を分離してパーティションの割り当てを効率化する。

2. 関連研究

2.1 キャッシュパーティショニング

マルチコア、SMT プロセッサの共有キャッシュにおいてコア間、スレッド間の競合を回避する方式としてキャッシュパーティショニング方式が提案されている。多くの方式では共有キャッシュを各コアに対して way 単位²⁾³⁾⁴⁾⁵⁾、または一定の領域単位⁶⁾⁷⁾ で配分する。

Suh ら⁷⁾²⁾ は SMT プロセッサ、マルチコアプロセッサの共有キャッシュに対してキャッシュミスを最小化するパーティショニング方式を提案している。この方式ではハードウェアカウンタを追加してキャッシュ領域毎にヒット数を求め、領域単位で最適なパーティショニングを求める。しかしこれらの方式はキャッシュの領域単位、way 単位でモニタリングを行うための多量のハードウェアが必要な点が問題である。

これらの方式に対して Qureshi ら³⁾ はモニタリングに必要なハードウェア量を削減する Utility-Based Cache Partitioning (UCP) を提案している。UCP はキャッシュ内の一部のセットのみをモニタリングしてパーティショニングを行うことでハードウェア量を削減する。また Dybdahl ら⁴⁾ もモニタリングに必要なハードウェア量を削減した Cache-Partitioning Aware Replacement Policy (CPARP) を提案している。CPARP はキャッシュの各セット

^{†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

^{†2} NEC システムプラットフォーム研究所

NEC System Platform Research Laboratories

に対して shadow tag を追加し、モニタリングを一部の way のみで行うことでハードウェア量を削減する。一方 Nikas ら⁵⁾ は Bloom Filter を用いてパーティション変更時のミス数を予測する Adaptive Bloom Filter Cache Partitioning (ABFCP) を提案している。

キャッシュパーティショニングはキャッシュを共有するコアに対してミスを最小化するように容量を割り当てる。しかしパーティションの決定において、各コアが使用する個々のブロックのアクセス頻度は考慮されていない。そのためキャッシュの全領域を各コアのパーティションに割り当てる。つまり各コアに必要以上のキャッシュ容量が割り当てられることで、パーティション内にアクセス頻度の低い dead block が含まれる点が問題である。またキャッシュパーティショニングは、アクセスのモニタリングに多量のハードウェアが必要な点が問題である。

2.2 dead block の排除

キャッシュミスの増加、利用効率の低下の原因となる dead block を優先的に排除する方法の一つとして、Cache bypassing⁸⁾⁹⁾ が提案されている。Cache bypassing は dead block となるデータをキャッシュに格納しないことで競合ミスを削減する。

dead block を排除するもう一つの方法として、LRU より正確に dead block を予測して優先的に dead block を置換する方式が複数提案されている¹⁰⁾¹¹⁾¹²⁾。また Kaxiras ら¹³⁾ は dead block を予測して競合を回避する代わりに電力供給を停止することで省電力化を図る Cache decay を提案している。

マルチコアプロセッサの共有キャッシュにおいて dead block の排除を行う方式として Jaleel ら¹⁴⁾ は Dynamic Insertion Policy (DIP)¹⁵⁾ を共有キャッシュに適用した Thread-Aware DIP (TADIP) を提案している。DIP は新しいデータの挿入位置を LRU と MRU からランダムに選択する Bimodal Insertion Policy (BIP) と LRU からミスが少なくなるポリシーを選択する方式である。さらに Jaleel ら¹⁶⁾ はキャッシュラインの再アクセス時間を予測して dead block を予測、置換する Re-Reference Interval Prediction (RRIP) を提案している。

これらの方式は直接的に dead block を予測、排除することができる。しかし dead block を予測するためには、ブロック毎のアクセス順序以外にアクセスパターンを記録するためのアクセス頻度、アクセス時期の状態の記憶領域が必要である。またブロック毎の記憶領域の容量は、dead block の予測アルゴリズムが検出するアクセスパターンの複雑性に比例して増加する問題を持つ。

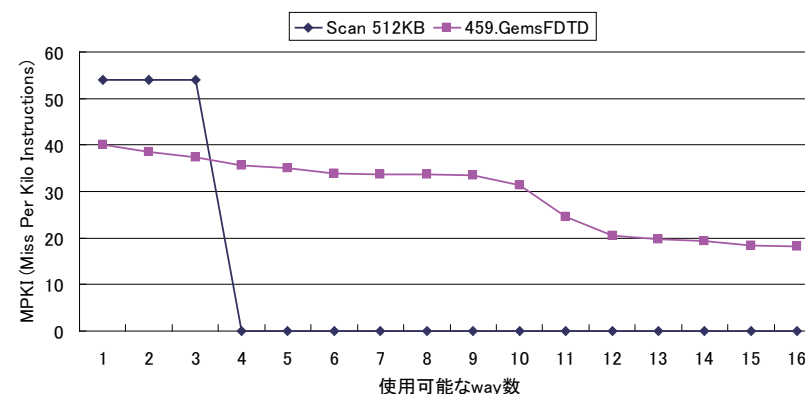


図 1 scan アクセスと使用可能なキャッシュの way 数の関係

3. 研究動機

マルチコアプロセッサの共有キャッシュでは、各コアによるアクセス、及び各コアの dead block の影響よりコア間の競合が発生する。そのため各コアが占有可能なキャッシュの容量は他のコアで実行中のプログラム、及びその動作状態によって変化する。また各コアが占有可能なキャッシュ容量が多い場合に置換が不要なブロックに対して、キャッシュ容量が少ない場合は置換が必要になる。

つまり dead block は他のコアの動作状態によって変化する。具体例として scan と呼ばれるアクセスパターンを用いて説明する。scan は連続した一定サイズの領域に対する断続的なアクセスの繰り返しである。図 1 に 512KB のデータに対する scan アクセスを行うプログラム、及び SPEC CPU20006 の GemsFDTD について、使用可能なキャッシュの容量とキャッシュミス頻度の関係を示す。図 1 の横軸は使用可能なキャッシュの way 数を表しており、1 way あたり 128KB の容量である。縦軸は 1000 命令あたりのキャッシュミスの頻度 (MPKI: Miss Per 1000 Instructions) を表している。

512KB の scan アクセスを行うプログラムに対して scan 対象のデータを格納するためには 4 way のキャッシュが必要である。図 1 の結果ではキャッシュを 4 way 以上使用可能な場合にミスが減少する。つまり 4 way 以上のキャッシュが占有可能である場合、512KB の scan データを格納したブロックは次にアクセスされるまでキャッシュ上に残るため、dead

block とはならない。一方使用可能な way 数が 3 以下である場合、512KB の scan データを格納したブロックは次にアクセスされるまでの間に、他のデータに置換される dead block に変化する。よってキャッシュパーティショニングにおいて、キャッシュ容量の変化に応じて dead block を判定し、各コアのパーティションの容量を決定する必要がある。同様に GemsFDTD についても、占有可能な way 数が 10 から 12 に増加することでキャッシュミスの頻度が減少している。これは scan アクセスを行うプログラムと同様に、キャッシュの容量が増加したことで dead block となっていたブロックの置換が不要になったためであると推測できる。

また従来の多くのキャッシュパーティショニング方式では、キャッシュミスの最小化を目的として各パーティションの容量を決定する。そのためパーティションに含まれる各ブロックの利用効率が考慮されず、dead block として各パーティションに含まれるブロックが含まれる。パーティションの割り当てを各コアからのアクセスに応じて行うことでこれらの dead block となるキャッシュ容量の割り当てを防ぎ、共有キャッシュの利用効率を向上できる。

本論文ではマルチコアプロセッサの共有キャッシュにおいて我々が提案したキャッシュパーティショニング方式である HFCA¹⁾ (History-Free Cache Allocation) において各コアに対する dead block の割り当てを回避する方式を提案する。HFCA は共有キャッシュを各コアがアクセスするデータを格納する占有パーティションと、dead block を含んだ共有パーティションに分離する。HFCA により少量のハードウェアでキャッシュパーティショニングを行い、dead block を分離し、割り当てを回避する。

4. パーティショニング方式 HFCA による dead block の排除

4.1 HFCA によるパーティショニングの基本原則

HFCA は他の多くのパーティショニング方式とは異なり、各コアのキャッシュヒットを監視して way 単位の粒度でパーティションを段階的に拡張する。そのため共有キャッシュは各コアが占有可能なパーティション (Private Partition: PP) と全コアから共有されるパーティション (Shared Partition: SP) に分割される (図2)。共有キャッシュが LRU 置換ベースの set-associative キャッシュであれば、各コアの占有する PP は共有キャッシュの MRU 側に配置し、SP を LRU 側に配置することで MRU 側に存在する PP にはキャッシュヒットしたデータが格納され、LRU 側の SP には PP から追い出されたデータが格納される。

よって各コアの PP と SP は、共有キャッシュの内部に仮想的な階層キャッシュを構成する (図3)。つまり PP は共有キャッシュ内で、各コアが占有する仮想的な L1 LRU キャッ

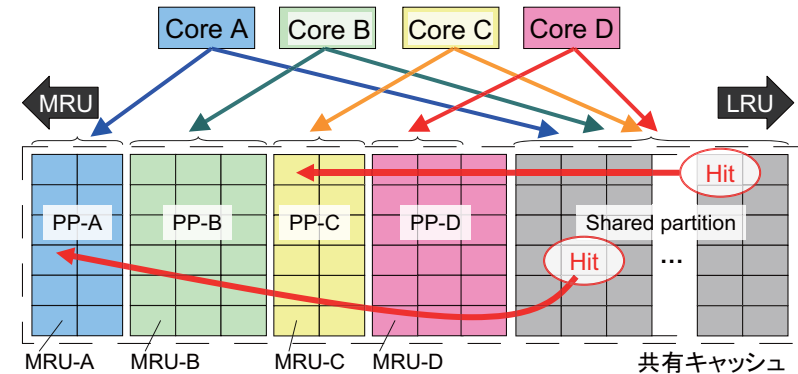


図2 共有キャッシュの Private partition と Shared partition への分割

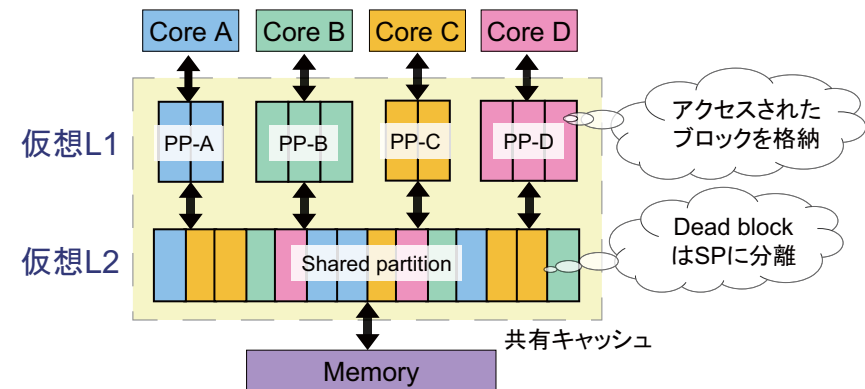


図3 HFCA による共有キャッシュ内の仮想的な階層構造

シュとして動作する。一方 SP は共有キャッシュ内で仮想的な共有 L2 キャッシュとして動作し、仮想 L1 キャッシュである各コアの PP から追い出されたブロックを格納する。逆に SP でヒットしたキャッシュブロックは仮想 L1 キャッシュである PP に格納される。HFCA は SP における各コアのキャッシュヒットを監視して、各コアの PP の way 数を変更することでコア間の競合ミスを回避する。

さらに HFCA では共有キャッシュを仮想的に階層化することで、PP に各コアが使用するブロックを格納し、アクセスされない dead block、若しくはアクセス頻度の低いブロック

を SP に分離する。SP に分離された dead block は PP に対して影響を与えず、キャッシュヒットしたブロックは各コアのパーティションの容量変更によって PP に格納される。これによってコア間のキャッシュ競合の回避と dead block の排除の両方を実現する。

4.2 各コアのパーティションの way 数の決定

HFCA は PP に各コアが使用するデータを格納することで、コア間の競合ミスを削減する。そのため HFCA は SP のキャッシュヒットを監視して、各コアの PP を way 単位で増減する。SP でキャッシュヒットが発生した場合、他のブロックを PP から追い出さずにヒットしたブロックを格納するために PP の way 数を 1 増やし、SP の way 数を 1 減らす。各コアの PP に対する way 数増加はキャッシュヒットが発生した順番に、SP の way 数が 0 になるまで行われる。つまり SP におけるヒットの頻度が高いコア、即ちワーキングセットが大きくアクセス頻度の高いコアの PP に対して way の割り当てが優先的に行われる。

HFCA における各コアの PP の way 数は、SP におけるキャッシュヒットによって決定される。そのため多くのパーティショニング方式とは異なりキャッシュアクセスをモニタリングし、履歴を記録するためのハードウェアが不要である。

一方各コアが使用しなくなった PP の way を解放して他のコアが利用可能な状態とするために、HFCA は一定時間毎に各コアの PP の way 数を削減する。本論文において HFCA は共有キャッシュで 50 万回アクセスが発生する毎に各コアの PP の way 数を半分 (小数点以下切り捨て) に削減する。

4.3 HFCA におけるキャッシュ置換ポリシー

HFCA では各コアに対するパーティショニングを行うため、各ブロックに最後にアクセスしたコアの情報が必要である。また各コアに割り当てられたパーティション以外のキャッシュ領域について特定のコアによる占有を防ぎ、公平に利用する置換ポリシーが必要である。

HFCA は各コアが使用するブロックに対して LRU で置換対象のブロックを選択する。HFCA は新たなデータを格納するブロックを、格納先のセットにおいて PP 以外のブロックを最も多く占有するコアから選択する。そしてブロックの置換対象となったコアが占有する各ブロックから LRU のブロックを実際に置き換える。格納先のブロックは SP の way 数が 1 以上の場合は SP に含まれる。一方 SP の way 数が 0、つまりキャッシュの全領域が各コアのパーティションとして割り当てられている場合、ブロックを置換するブロックが占有する LRU のブロックを格納先のブロックに選択する。

選択された新しいデータを格納するブロックに対して、次に置換順序の情報を更新する。キャッシュを効率的に利用するためには、dead block を優先的に排除するだけでなく dead

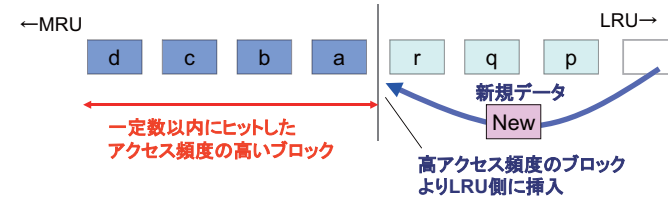


図 4 HFCA による共有キャッシュへの新規ブロックの挿入位置

表 1 プロセッサコアとキャッシュの設定

コアの構成	
コア数	2
Instruction set	MIPS
L1 cache	Instruction/Data 独立 32KB/2-way/LRU 置換 line size 32B/1-cycle latency
キャッシュのバンク・メモリの構成	
Cache bank	Instruction/Data 共有 2MB/16-way/line size 32B/11-cycle latency
Memory	300-cycle latency

block から受ける影響を回避する必要がある。そのため HFCA では新たにデータが格納されたブロックの置換の優先度を MRU 側ではなく、アクセス頻度の高いブロックに比べて LRU 側に配置されるように変更する (図 4)。これによって選択されたブロックが 1 度しかアクセスされず dead block となった場合に他のアクセス頻度の高いブロックを置換することを防ぐ。本論文では、HFCA は共有アクセスにおいて 1 万回以内にヒットしたブロックを格納するために必要な way 数を記録し、新たなデータを格納したブロックがその way 数より LRU 側 (way 数が各コアのパーティションに等しい場合は LRU) になるように挿入する。

5. 性能評価

5.1 評価環境

本論文で提案した HFCA を用いたキャッシュパーティショニングによるキャッシュ割り当て、及び dead block の排除効果について、SESC¹⁷⁾ シミュレータ上で scan アクセスを行うプログラム実行時の性能を評価する。HFCA による dead block 排除の効果 (以下、HFCA) を、dead block の排除を行わない通常の LRU キャッシュを用いた場合 (以下、LRU) の性能と比較する。本論文の性能評価では、各コアの IPC の合計値をプロセッサ全体の IPC と

して評価指標に用いる。また dead block の排除効果として、HFCA が共有キャッシュ内で各コアにパーティションとして割り当てたキャッシュの way 数を用いる。

scan アクセスは共有キャッシュに格納可能な 512KB、及び格納不可能な 4MB の連続したメモリ領域に対して行う。scan アクセスを行うプログラムと並列して、マルチコアプロセッサ上の他のコアで SPEC CPU2006 を実行する。SPEC CPU2006 は specrand の 2 種類を除いたプログラム、入力データは ref を用いる。各コア上のプログラムは、それぞれ先頭 5G 命令をスキップ後に 500M 命令を実行して評価を行う。

コア及びキャッシュに関するシミュレーションパラメータは特に断りのない限り表 1 に示した値を使用する。L2 キャッシュは 2 コアで共有されており、ラインサイズ 32 byte, 16 way の LRU キャッシュである。

5.2 小容量の scan アクセスに対するキャッシュ割り当て

scan アクセスの容量が共有キャッシュの容量より小さく way 数の不足が生じない場合、scan アクセスされた全てのデータをキャッシュに格納できる。つまり並列実行されるプログラムによる場合を除いて、scan アクセスされたデータの置換は発生せず、HFCA による dead block の分離は性能向上効果を持たない。本論文の評価では 512KB の scan アクセスと SPEC CPU 2006 の各プログラムを並列に実行する場合に、HFCA の適用によりプロセッサ全体の IPC が低下する結果が得られた。しかしながら IPC の低下幅は 0.1% 未満であり、性能への影響は微小である。

次に HFCA による dead block の削減効果を評価するために、SPEC CPU2006 の各プログラム実行時に HFCA が各コアにパーティションとして割り当てた共有キャッシュの way 数を図 5 に示す。横軸は scan アクセスと並列して実行した SPEC CPU2006 の各テスト、縦軸は 16 way の共有キャッシュ全体で割り当てた平均 way 数を表す。scan アクセスは共有キャッシュの容量の 2MB より小さい 512KB である。

全テストにおいて割り当てられた平均 way 数は 11.2 であり、キャッシュ全体の平均約 70% である。つまり HFCA は平均で約 30% の way に dead block が格納されていると認識し、Shared partition の形で分離をしている。特に milc、libquantum と scan アクセスを並列実行した場合は、HFCA が割り当てを行う平均 way 数はキャッシュ全体の 36% に相当する 5.8 まで抑えられる。また 512KB の scan アクセスのデータを全て共有キャッシュ上に格納するためには 4 way 分の領域が必要である。これに対して図 5 の結果から、scan アクセスを行うコアに対していずれのテストにおいても約 4 way のパーティションが割り当てられていることが分かる。

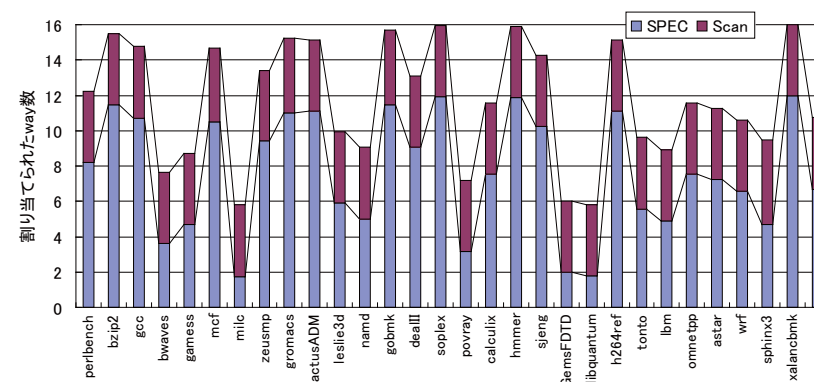


図 5 SPEC CPU2006 と 512KB の scan アクセスの並列実行で割り当てられた平均 way 数

つまり HFCA は小容量の scan アクセスに対して、キャッシュ可能であることを正しく判定可能であることが分かる。またそれらのアクセスを行うコアに対して大幅な性能低下を伴わずにキャッシュミスの最小化に必要なパーティションの割り当て、及び dead block の排除効果を同時に実現できることが分かる。

5.3 大容量の scan アクセスに対するキャッシュ割り当て

続けてキャッシュへの格納が不可能な、4MB の scan アクセスに対する HFCA のキャッシュの割り当て、及び dead block 排除の効果を評価する。まず SPEC CPU2006 の各テストと 4MB の scan アクセスの並列実行における、HFCA 適用時の IPC を図 6 に示す。横軸は SPEC CPU2006 の各テスト、縦軸は HFCA 非適用時の各コアの IPC の合計値を 1 とし正規化した HFCA 適用時の IPC を表す。HFCA の適用により、IPC は SPEC CPU2006 全体で平均約 6.5% 向上しており、HFCA が平均的に IPC の向上に効果を持つことが分かる。

また bzip2、soplex、hmmmer、xalanbmk に対して HFCA は 15% 以上の高い IPC 向上効果を示している。これらのプログラムが占有可能な way 数とキャッシュミス頻度の関係を図 7 に示す。横軸は占有可能なキャッシュの way 数を表しており、縦軸はキャッシュミスの頻度 (MPKI: Miss Per Kilo Instructions) を表す。図 7 の結果から、HFCA による IPC 向上効果が高かった SPEC CPU2006 のテストは最大 way 数まで占有可能な way 数を増やすことで、キャッシュミスの頻度が下がることが分かる。つまりこれらのプログラムでは再

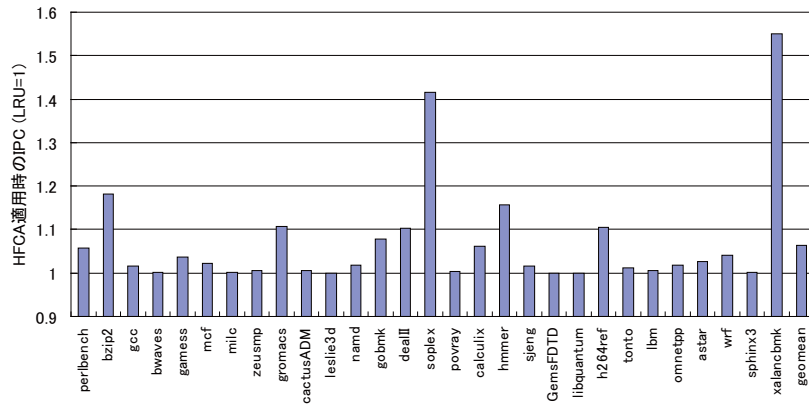


図 6 SPEC CPU2006 と 4MB の scan アクセスの並列実行に対する IPC 向上効果

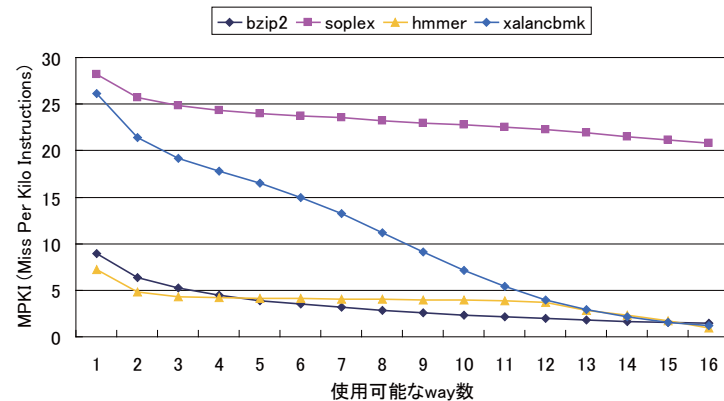


図 7 IPC 向上効果の大きいプログラムにおける使用可能な way 数とキャッシュミス頻度の関係

アクセスされるブロックが多いため、scan アクセスによるブロックの置換の影響を受けやすく、HFCA によるパーティショニング、及び dead block 排除の効果が大きいと推測できる。一方で lesie3d, libquantum において HFCA の適用により IPC の低下が発生している。しかし IPC の低下は最大で約 0.1%と微小であり、SPEC と大容量の scan アクセスの

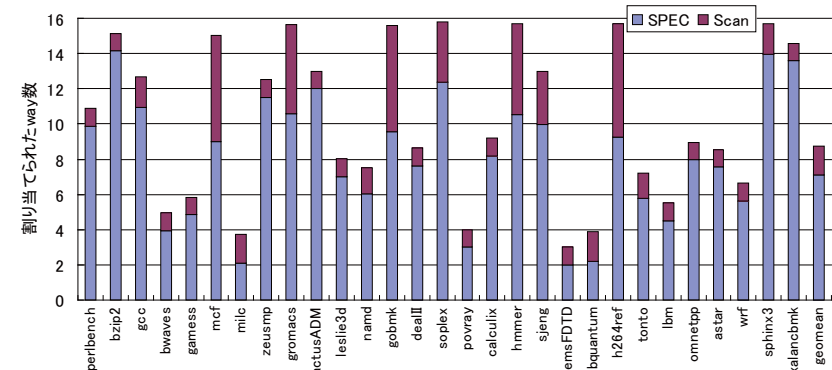


図 8 SPEC CPU2006 と 4MB の scan アクセスの並列実行で割り当てられた平均 way 数

並列実行下において HFCA による IPC の低下は発生しないと推測できる。

次に HFCA による dead block の削減効果を評価するために、SPEC CPU2006 の各プログラム実行時に HFCA が各コアにパーティションとして割り当てた共有キャッシュの割合を図 8 に示す。横軸は scan アクセスと並列して実行した SPEC CPU2006 の各テスト、縦軸は 16 way の共有キャッシュ全体で割り当てた平均 way 数を表す。scan アクセスは共有キャッシュの容量の 2MB より大きい 4MB である。

全テストにおいて割り当てられた平均 way 数は 9.1 way であり、キャッシュ全体の平均約 57%である。つまり HFCA は平均で約 43%の way に dead block が格納されていると認識し、Shared partition の形で分離をしている。また 4MB の scan アクセスを行うコアに割り当てられた way 数の平均は約 1.6 way である。HFCA は各コアに最低 1 way を割り当てるため、多くのテストとの並列実行時にキャッシュヒットさせることが困難な scan アクセスに対する way の割り当てが行われていないと推測できる。つまり scan アクセスによって生じる dead block を正しく排除していることが分かる。

よって HFCA はキャッシュへの格納が不可能な大容量の scan アクセスに対して、ミスをも最小化するパーティションの割り当て、及び dead block の排除効果を同時に実現できることが分かる。しかし一部のテストでは平均で 2 way 以上が scan アクセスに割り当てられている。その原因は、HFCA が新たにデータが格納されたブロックの置換の優先度をアクセ

ス頻度の高いブロックに比べて LRU 側に配置するためであると推測できる。

6. ま と め

本論文では我々の提案したマルチコアプロセッサ向けの共有キャッシュパーティショニング方式である HFCA¹⁾を用いて共有キャッシュから dead block を排除し、各コアに効率的にキャッシュを割り当てる方式を提案した。HFCA は各コアのキャッシュヒットに応じて、ヒットしたブロックを含むように各コアにパーティションを way 単位で割り当て、各コアのパーティションには再アクセスされたブロックを格納する。HFCA は各コアに対するパーティションの割り当てを繰り返すことで、dead block を共有キャッシュ内の共有パーティションに分離し、dead block による競合、パーティション割り当てへの影響を防ぐ。本論文の評価の結果、2MB の共有キャッシュを持つマルチコアプロセッサにおいて、SPEC CPU2006 と 4MB の scan アクセスの並列実行に対して HFCA の適用で IPC が平均 6.5% 向上し、パーティションとして割り当てる way 数がキャッシュ全体の 57% に削減されることを確認した。つまり HFCA により少量のハードウェアで dead block を認識し、各コアのキャッシュミスを防ぐために必要なパーティションの割り当てが可能であることが判明した。以上の評価からキャッシュミスを最小化するパーティションの割り当て、及び dead block の排除効果を同時に実現する点において HFCA の有用性を示した。

参 考 文 献

- 1) 小川周吾, 入江英嗣, 平木 敬: アクセス履歴の不要なマルチコア CPU 向け共有キャッシュ配分方式, 先進的計算基盤システムシンポジウム SACSIS2010, pp.267-276 (2010).
- 2) Suh, G., Rudolph, L. and Devadas, S.: Dynamic Partitioning of Shared Cache Memory, *Journal of Supercomputing*, Vol.28, No.1, pp.7-26 (2004).
- 3) 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).
- 4) 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).
- 5) Nikas, K., Horsnell, M. and Garside, J.: An adaptive bloom filter cache partitioning scheme for multicore architectures, *Proc. International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp.25-32 (2008).
- 6) Stone, H.S., Turek, J. and Wolf, J.L.: Optimal Partitioning of Cache Memory,

IEEE Transactions on Computers, Vol.41, No.9, pp.1054-1068 (1992).

- 7) 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).
- 8) Tyson, G., Farrens, M., Matthews, J. and Pleszkun, A.R.: A modified approach to data cache management, *Proc. 28th annual international symposium on Microarchitecture*, pp.93-103 (1995).
- 9) Johnson, T.L., Connors, D.A., Merten, M.C. and mei W.Hwu, W.: Run-Time Cache Bypassing, *IEEE Transactions on Computers*, Vol.48, No.12, pp.1338-1354 (1999).
- 10) Lai, A.-C., Fide, C. and Falsafi, B.: Dead-block prediction & dead-block correlating prefetchers, *Proc. 28th Annual International Symposium on Computer Architecture*, pp.144-154 (2001).
- 11) Liu, H., Ferdman, M., Huh, J. and Burger, D.: Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency, *Proc. 41st Annual International Symposium on Microarchitecture*, pp.222-233 (2008).
- 12) Xiang, L., Chen, T., Shi, Q. and Hu, W.: Less Reused Filter: Improving L2 Cache Performance via Filtering Less Reused Lines, *Proc. 23rd ACM International Conference on Supercomputing*, pp.68-79 (2009).
- 13) Kaxiras, S., Hu, Z. and Martonosi, M.: Cache decay: exploiting generational behavior to reduce cache leakage power, *Proc. 28th Annual International Symposium on Computer Architecture*, pp.240-251 (2001).
- 14) Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Jr., S.S. and Emer, J.: Adaptive Insertion Policies for Managing Shared Caches, *Proc. 17th International Conference on Parallel Architecture and Compilation Techniques*, pp.208-219 (2008).
- 15) Qureshi, M.K., Jaleel, A., Patt, Y.N., Jr., S. C.S. and Emer, J.: Adaptive Insertion Policies for High Performance Caching, *Proc. 34th Annual International Symposium on Computer Architecture*, pp.381-391 (2007).
- 16) Jaleel, A., Theobald, K., Jr., S. C.S. and Emer, J.: High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP), *Proc. 37th Annual International Symposium on Computer Architecture*, pp.60-71 (2010).
- 17) Renau, J., Fraguera, B., Tuck, J., Liu, W., Prvulovic, M., Ceze, L., Sarangi, S., Sack, P., Strauss, K. and Montesinos, P.: SESC simulator (2005). <http://sesc.sourceforge.net>.