Vol.2011-ARC-196 No.10
2011/7/27

CMP 向け分散キャッシュにおけるキャッシュパーティショニング方式

藤枝直輝[†1]　吉瀬謙二[†1]

プロセッサのシングルスレッド性能向上の限界により，プロセッサに搭載されるコアの数は増加傾向にある．このようなチップマルチプロセッサ (CMP) のキャッシュの利用法として，コアごとに分割されたラストレベルキャッシュ(LLC) を持ち，あるコアのキャッシュから追い出されたラインを，別のコアのキャッシュにおける利用率の低い領域へと移動させることで，その容量を柔軟に利用する方法が提案されている．本稿では，このような分散キャッシュに対して，RFF(Reference Frequency Filter) を改良した機構を利用し，キャッシュの利用率が低い領域を高速かつ正確に予測することで，キャッシュの効率を高められることを明らかにする．評価の結果，8 コアで 2 つの並列アプリケーションを同時に動作させた場合，オフチップアクセスの頻度は平均 5.4%，アプリケーションの組み合わせによっては最高で 29.7%減少し，プロセッサの性能向上の可能性があることを確認した．

The cache partitioning method for CMP cooperative caching

Naoki Fujieda[†1] and Kenji Kise[†1]

The limitation of single thread performance leads processor architecture to increase the number of cores. To use the cache capacity of chip multiprocessor (CMP) efficiently, recent technique proposed to have core-divided last-level cache (LLC) and forward evicted lines from one cache to rarely used region of another cache. In this paper, we apply a modified version of RFF (Reference Frequency Filter) to LLC partitioning unit. And we clarify that it enables a fast and accurate detection of rarely used region and improves the processor performance. Our evaluation shows that the reduction of the number of off chip access over the base partitioning scheme is 5.4% on average and 29.7% at a maximum.

1. はじめに

近年では，プロセッサ内に複数のコアを搭載し，スレッドレベル並列性を利用してプロセッサのスループットを向上させる CMP (Chip Multiprocessor) が主流となっている．また，プロセッサの速度向上に対して主記憶の速度向上が小さいために，主記憶のアクセスレイテンシは相対的に増加している．そのため，CMP においてはオフチップの主記憶アクセスが性能を制限する大きな要因となり，キャッシュの利用効率を高めることが重要な課題となっている．

大容量・単一のキャッシュは単一のアクセスレイテンシをもつが，容量が大きくなるほどそのレイテンシは大きくなってしまう．これを改善するため，CMP では比較的小容量のキャッシュを複数並べた NUCA 型のキャッシュ構成がしばしば用いられる[1),2)]．NUCA ではアクセスレイテンシは不均一となるが，データがアクセス元に近いキャッシュにヒットすれば，レイテンシを低く抑えることが可能である．

最近ではこのアプローチを発展させて，各コアごとに分割されたラストレベルキャッシュ (LLC) を持ち，あるコアのキャッシュから追い出されたラインを別のコアのキャッシュへと移動させることで，その容量を柔軟に利用する方法が提案されている[3),4)]．このようなキャッシュ構成においては，将来的に再利用される可能性が高いラインのみを移動の対象とし，その移動先を別のコアにおける再利用の可能性が低い領域へと限定することで，キャッシュの利用効率を向上できることが明らかになっている[5),6)]．このことを利用すると，大容量のキャッシュを必要とするアプリケーションと，キャッシュ容量をさほど必要としない，すなわちワーキングセットが十分小さいか，あるいはストリーミング処理のようにキャッシュの再利用率が極端に少ないアプリケーションとが混在するケースでは，前者のコアはあたかも大容量のキャッシュを持っているかのように高速に動作できる．また，キャッシュの再利用率が極端に少ないアプリケーションが存在するケースにおいては，こうしたアプリケーションが他のコアが必要とするラインを汚染し（追い出してしまい），性能を低下させるリスクを取り除くことが可能となる．こうした利点を最大限に享受するためには，キャッシュの利用率が低い領域を高速かつ正確に予測する必要がある．

本稿では，文献 6) で提案されている Elastic Cooperative Caching (ElasticCC) を対象

†1 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

Vol.2011-ARC-196 No.10
2011/7/27

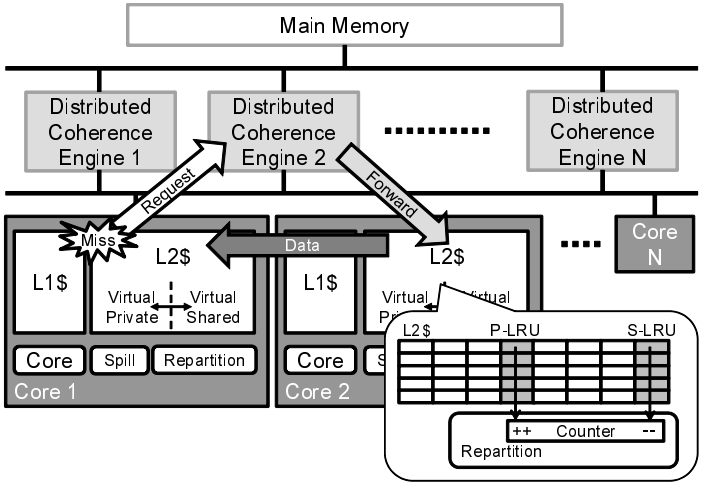

図 1 Elastic Cooperative Caching の構成と動作例.

に，キャッシュパーティショニングの改善方式を提案する．提案方式では，アクセス頻度の低いラインを検出する機構である RFF (Reference Frequency Filter)[7] に修正を加えた方式を適用し，キャッシュの利用率が低い領域を予測し，パーティショニングの指標とする．これによりキャッシュの利用効率を高め，プロセッサの性能向上を図る．

本稿の構成を述べる．2 章で提案方式のベースとなる ElasticCC について述べ，3 章で RFF，および提案方式のために修正した RFF の構成を説明する．4 章で提案方式をメニーコアプロセッサシミュレータに実装して，オフチップアクセスの頻度と実行時間を評価し，5 章で本稿をまとめる．

2. Elastic Cooperative Caching

図 1 に，本稿で提案する機構の適用先である Elastic Cooperative Caching (ElasticCC)[6] の構成を示す．

ElasticCC の特徴のひとつは，チップ内に分散されたコヒーレンス制御エンジン (Distributed Coherence Engine, DCE) を持つことである．これは ElasticCC のベースである，Distributed Cooperative Caching(DCC)[4] からの特徴である．各々の DCE は，チップ内全てのキャッシュのディレクトリのうち，アドレスでインタリーブされた一部分を持っており，アドレス空間の一部分のコヒーレンス制御を司っている．

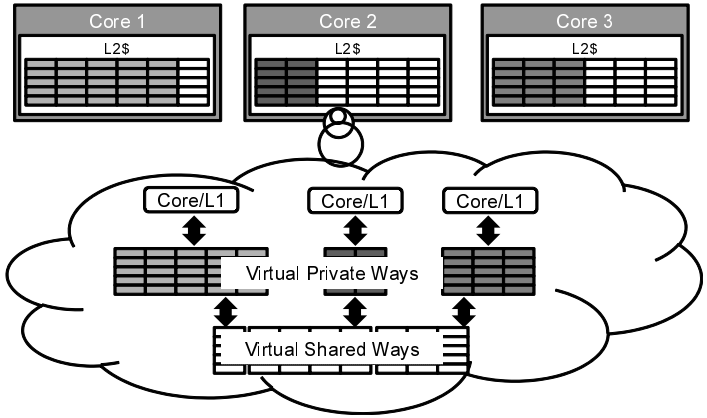

図 2 L2 による仮想的なキャッシュ階層.

あるコアが自分のコアのキャッシュにミスすると，要求はまず，アドレスによって一意に定められた DCE に転送される．DCE は要求されたラインの状態を調べ，別のコアのキャッシュ内に所望のデータがあることがわかればデータを持つコアへと要求を転送する．要求を受けたコアは，キャッシュ間転送により要求を発したコアへとデータを送信する．一方，要求されたラインがどのコアにも存在していなければキャッシュミスとなり，主記憶が要求を発したコアへデータを供給する．このときディレクトリに無効なエントリが存在しない場合は，適当なエントリを無効化して空きを作る必要がある．これは同時にそのエントリに対応したキャッシュラインが全て無効化されることを意味する．これを実現するために，ElasticCC のコヒーレンスプロトコルにはいくつかの追加の状態を必要とする．また，頻繁なディレクトリエントリの無効化は性能に悪影響を及ぼすので，キャッシュの総ライン数に対して，DCE の総エントリ数をある程度（典型的には 1.5〜2 倍に）大きく設定することもできる．

もうひとつの特徴は，各コアの L2 キャッシュが各コア占有 (Private) と全コア共有 (Shared) の領域をもつことである．各コアのキャッシュから追い出されたラインのうちは，1 回に限り別のコアのキャッシュへと転送されるが，これにより受信されたラインは共有領域の側へと格納される．すなわち，全コア共有の領域は図 2 に示すような仮想的な L3 キャッシュとして機能する．占有・共有の割合はウェイ単位で可変であり，キャッシュ内のリパーティ

ショニングユニット (図中の Repartition) が，必要ならば一定サイクルごとにキャッシュのパーティショニングを変更する．ただし，制御の簡単化のために占有・共有それぞれに必ず 1 ウェイ以上は割り当てられるものとする．すなわち，あるコアのキャッシュにおけるパーティショニングの選択は，(L2 キャッシュのウェイ数-1) 通り存在する．

ElasticCC のリパーティショニング制御は，図 1 の右下に示すインクリメント・デクリメント可能なカウンタを用いて行う．まず，占有領域と共有領域とのそれぞれで LRU(Least Recently Used) のウェイへのヒットを検出する．もし占有領域での LRU(図中の P-LRU) にヒットした場合にはカウンタをインクリメントし，共有領域での LRU(図中の S-LRU) にヒットした場合はカウンタをデクリメントする．リパーティショニングユニットは一定サイクルごとにカウンタの値を確認する．カウンタの値があらかじめ定められた上のしきい値よりも大きければ，占有領域がより多くのキャッシュ領域を必要としていることを意味するので，占有領域に割り当てるウェイ数を 1 つ増加させる．逆に，カウンタの値が下のしきい値よりも小さければ，共有領域に割り当てるウェイ数を 1 つ増加させる．そのいずれでもない，すなわちカウンタの値が上下のしきい値の間に位置しているならば，パーティションの変更は行わない．最後に，カウンタの値をリセットしてその回のリパーティショニングは終了となる．

ここでパーティションが変更された場合，その情報は（必要ならば優先度の低いチャネルを用いて）ブロードキャストされる．この情報は各コアのキャッシュの Spilled Blocks Allocator (図中の Spill) に伝えられる．キャッシュから追い出されたラインを転送する場合には，この情報により集められた各コアの共有領域の大きさの割合と同じ割合でもって転送先を選択する．

ElasticCC には，オプションとして Adaptive Spilling とよばれる，どのラインを別のコアに移動すべきかを適応的に判断する機構を追加できる．具体的には，自コアのキャッシュにおける占有領域の割合が 3/4 以上である場合，あるいは追い出されたラインが過去に他のコアと共有されていた場合（現在共有されている場合は除く）に限り，キャッシュラインを別のコアへと転送する．

3. キャッシュパーティショニング方式の提案

3.1 Reference Frequency Filter

RFF(Reference Frequency Filter)[7] は，キャッシュにおけるアクセス頻度の低いラインを検出する機構である．この機構は HCFA(History-Free Cache Allocation)[8] と呼ばれる共有キャッシュ配分方式を対象としているが，HCFA は先に述べた ElasticCC と同様に，共有キャッシュを占有・共有の領域へと仮想的かつ動的に分割するアプローチを取っている．そのため，RFF は ElasticCC との親和性も高いと考えられる．

RFF では，占有領域において MRU(Most Recently Used) 側から何番目のウェイまでが実際にアクセスされたかを記録するレジスタを持つ．このレジスタの値を a として，計測したキャッシュヒットの回数を n，1 回のキャッシュヒットが a ウェイ目よりも外にヒットする確率を p とおくと，n 回のキャッシュヒットが全て MRU から a ウェイ目以内にヒットする確率は $(1-p)^n$ で求められるので，n を増やせば増やすほど p を低く見積もることが可能となる．これを利用して，計測された a の値を占有領域のウェイ数として用いつつ，n を増減させて期待する p の値を変化させるのが，この機構のねらいである．

しかしながら，RFF を ElasticCC へそのまま適用するには大きく 2 つの問題点がある．ひとつは，占有領域において RFF が期待するヒットの割合が高すぎることである．例えば $n = 500$ と置いたとき，計測した a の値に対する信頼度を 50%と仮定した場合でさえ，p には 99.86%以上を期待していることになる．すなわち，RFF はアクセス頻度が十分に低いウェイのみを共有領域として用いる，非常に保守的な機構であるともいえる．しかし，ほとんど使用されていない領域にまで占有領域とすべきという誤った判断がなされることで，重大な機会損失を生み出す可能性もある．これを防ぐために n を小さく設定することも考えられるが，今度は標本誤差の増大により，a の値についてのばらつきが増大してしまう．これを改善するには，a の値はある程度保ったまま，あるウェイへの 1 度のアクセスではなく，複数回のアクセスにより初めてレジスタが更新されるようにするといった方式が考えられる．

もうひとつの問題は，一定回数のキャッシュヒットをパーティション判定のトリガーとしていることから，プログラムが L2 キャッシュヒットのほとんど発生しない部分へと移行すると，その変化への対応が遅れる，または対応ができない場合があることである．これはある程度長い時間が経過したときには強制的に更新を行うことにより対処可能である．

3.2 提案手法

図 3 に示す，本稿で提案するキャッシュパーティショニング方式は，こうした観察をもとに RFF を改善したものである．提案方式には，前回のパーティショニングからの合計のキャッシュヒット数 (Total Hit)，MRU を除く各ウェイのヒット数 (Way Hit)，他コアのキャッシュに一度追い出されたラインがヒットした回数 (Victim Hit) の 3 種類のカウンタを利用する．

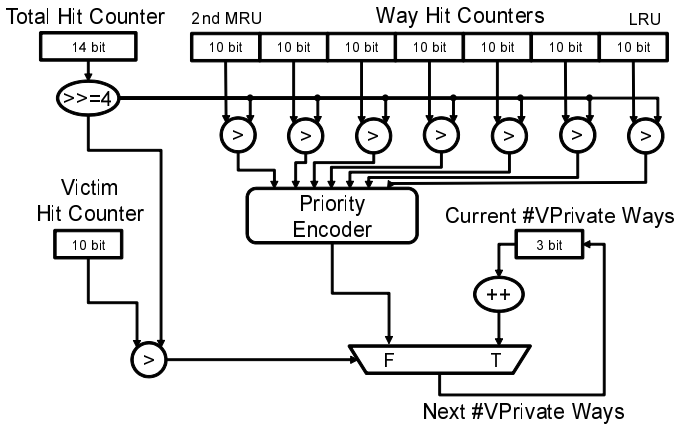

図 3 提案手法におけるパーティショニング機構の構成.

パーティションの変更は以下に示すとおりに行われる．まず，各ウェイのヒット数，および追い出されたラインのヒット数が，合計のヒット数の 1/16 よりも大きいかを判定する．判定結果は，LRU 側に高い優先度をもつ優先度付きエンコーダによって，全ての判定に失敗したら 1，MRU から 2 番目のウェイのみが判定に成功したら 2，というようにして，1〜(ウェイ数-1) の占有ウェイ数の候補へと変換される．もし追い出されたラインのヒット数判定に失敗していれば，この値が次回の占有ウェイ数となる．そうでなければ，現在の占有ウェイ数に 1 を加えたものを次回の占有ウェイ数として用いる．

パーティションの変更は，一定サイクルおきに合計のヒット数をチェックして，その値が一定のしきい値以上である場合にのみ行われる．長い間パーティションの変更が行われなかった場合，プログラムが L2 キャッシュヒットのほとんど発生しない部分へ移行したと判断し，占有ウェイ数を強制的に 1 に設定する．以下では，チェックの間隔を 50K サイクル，しきい値を 128，パーティションの強制変更までの期間を 500K サイクルと指定している．

追い出されたラインのヒット検出，すなわち，他のコアから受信したラインが過去に追い出されたラインかどうかを判定するには，ラインの送信元であるキャッシュのステートを調べればよい．ElasticCC では，前述のとおり追い出されたコアの別のコアへの移動は，ラインごとに 1 回だけに限られている．この判定のために，ElasticCC のキャッシュラインには，既に 1 度追い出されたラインかどうかを示す 1 ビットのフラグが既に用意されている．このフラグの情報を利用するので，追い出されたラインのヒット検出にはキャッシュラインへの追加の記憶容量なしに行える．具体的には，他のコアからの要求に対してラインのデータを返す際に，このフラグも併せて転送するようにする．ラインを受信したコアのキャッシュはフラグの情報を読み取って，フラグが有効であれば追い出されたラインのヒットとして扱う．

また，提案手法では Adaptive Spilling の条件についても変更を加えている．自らのキャッシュが 3/4 以上占有領域にある場合と，追い出されたラインが過去に他のコアと共有されていた場合にラインの移動を行うのは同じであるが，そうでない場合についても，1/32 の割合でラインの移動を行う．これは，追い出されたラインの潜在的なヒット数を測定するためのサンプルとして利用される．このライン移動の割合を考慮して，自らのキャッシュの占有領域の割合が 3/4 に満たない場合においては，追い出されたラインのヒット数に 4 倍の傾斜を持たせている．すなわち，1 回のヒットにつき Victim Hit カウンタを 4 ずつインクリメントする．

種類	パラメータ	値
計算ノード	プロセッサコア数	8
	プロセッサ	インオーダ，L1 ヒット時の IPC=1
	ISA	MIPS32
キャッシュ	ブロックサイズ	64B
	L1 I/D キャッシュ	各 8KB，2-way
	L2 キャッシュ	256KB, 8-way, 10 cycles
	DCE エントリ	6144 エントリ, 12-way
	コヒーレンスプロトコル	MOESI をもとに変更
主記憶	レイテンシ	200 cycles
ネットワーク	トポロジ	2 次元メッシュ
	ルータ	5 入力 5 出力，2 仮想チャネル
		X-Y 次元順ルーティング
	ホップレイテンシ	3 cycles
	リンク幅	16B/cycle

表 1 評価パラメータ.

4. 評　　価

4.1 評価環境

提案手法の性能評価には，本研究室で開発しているメニーコアプロセッサシミュレータである SimMc[9] に対し，キャッシュ，ディレクトリその他の拡張を施したものを用いる．表 1 に，主要な評価パラメータの一覧を示す.

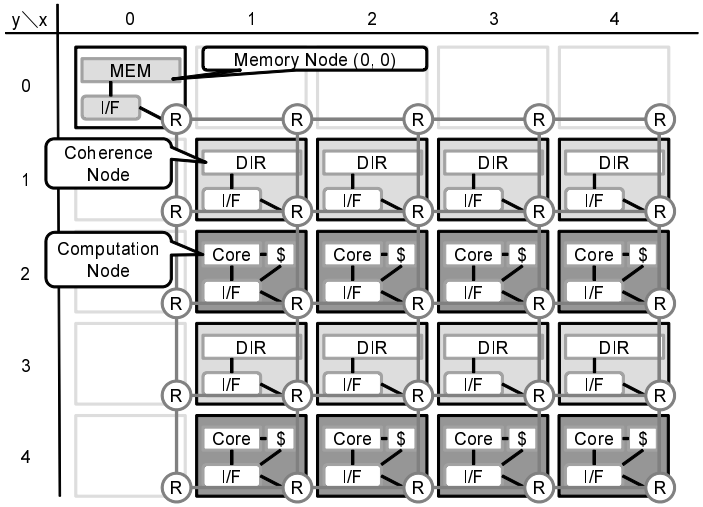

図 4 評価対象となるメニーコアの構成.

図 4 は，評価対象となるメニーコアの構成である．このメニーコアは 5 × 5 の 25 ノードの構成を取り，各ノードはルータによってメッシュ状に接続されている．ノードのうち座標 (0,0) のノードが主記憶にアクセス可能なメモリノード，(1,1)〜(4,1)，(1,3)〜(4,3) の計 8 ノードは DCE に相当するコヒーレンスノード，(1,2)〜(4,2)，(1,4)〜(4,4) の計 8 ノードはコアおよび L1，L2 キャッシュを持つ計算ノードであり，残りのノードはルータだけを持つパスノードである．計算ノードのうち，左側 4 つ (x=0,1) が第 1 のアプリケーションを，右側 4 つ (x=2,3) が第 2 のアプリケーションを，それぞれ並列に実行する．DCE は，キャッシュタグをコヒーレンスノードの個数，すなわち 8 で割った剰余によりインタリーブされる．なお，文献 4)，6) では，各々の DCE は 1 つのコアとともにルータに接続される構成をとっている（すなわち，ルータが 6 入力 6 出力となる）が，本稿ではルータの簡単のためにコアと DCE とを別ノードに離して配置している．

評価では，ElasticCC に Adaptive Spilling を追加したもの (ElasticCC+AS) を従来手法・比較のベースラインとして，キャッシュパーティショニング機構に提案手法を用いたものについて，オフチップアクセスの削減率，および性能の向上率を計測する．ここで性能の向上率は，2 つのアプリケーションの実行時間の短縮の割合について調和平均をとったものと定義する．ベースラインのキャッシュパーティショニング方式におけるパーティション変更のチェックは 100K サイクルごとに行う．また，アプリケーションのオフチップアクセス・性能の計測からは必要なデータの初期化にかかる部分は除外されている．

名前	内容	主なパラメータ	並列化
dijkstra	最短経路探索	128 ノード	タスク並列
equation	equation solver kernel[10]	384×384 要素	データ並列
himeno	姫野ベンチマーク	サイズ XS	データ並列
mm	行列積	128×128 要素	データ並列
qsort	クイックソート	160K 要素	データ並列

表 2 評価に用いるベンチマーク.

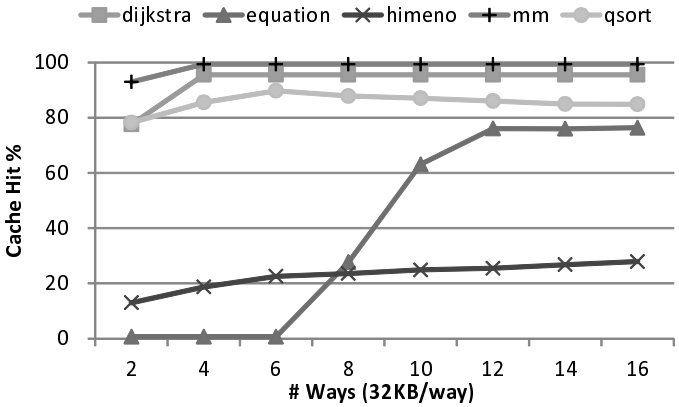

図 5 評価に用いるベンチマークのキャッシュ利用特性.

ベンチマークには表 2 で示す 5 種類を用いる．これらは全て 4 コアでの実行に合わせて並列化を施している．これら 5 種類のベンチマークの中から重複を許して 2 つ選択し，評価のための 1 つのセッティングを構成する．セッティング名は，選択した各々のベンチマークの頭文字を並べたものとする．例えば，dijkstra と qsort の組み合わせはセッティング D-Q と呼ぶ．ここで，2 つのアプリケーションの左右を入れ替えただけのものは同一のセッティングとしてカウントする．したがって，セッティングの数は合計 15 種類となる．

評価に用いるベンチマークにおけるキャッシュの利用特性は，図 5 に示す通りである．これは，L1 サイズを固定し，L2 サイズの容量およびウェイ数を増減させた場合の，キャッシュヒット率の変化についてまとめたものである．横軸にはウェイ数（すなわち容量），縦

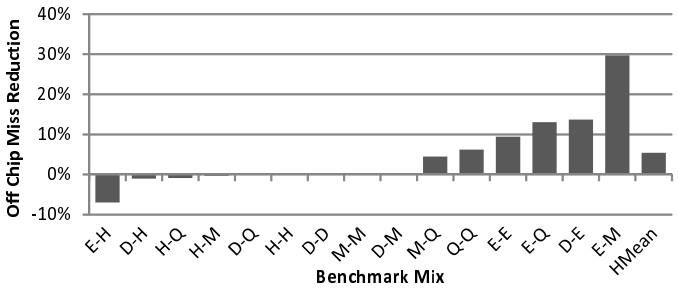

図 6 オフチップアクセスの削減率.

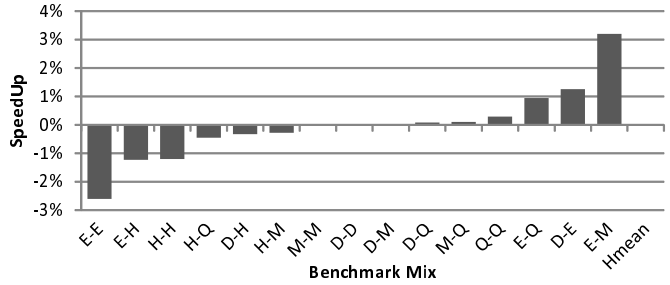

図 7 性能の向上率.

軸は L2 キャッシュヒット率である．dijkstra および mm は，ワーキングセットのサイズが L2 キャッシュに十分収まるほど小さいベンチマークである．qsort はワーキングセットのサイズは L2 キャッシュより大きいものの，データがすぐに再利用される割合が比較的高い．himeno はワーキングセットのサイズが L2 キャッシュよりもずっと大きく，キャッシュサイズの増加に対して緩やかにヒット率が増加していく．equation はワーキングセットのサイズが L2 キャッシュよりも大きく，連続的なデータの再利用も少ないが，周期的にアクセスされるデータが多く含まれており，キャッシュサイズを増加させていくとある点から急激にヒット率が上昇する．

4.2 キャッシュ利用効率および性能

前述した全 15 種類のセッティングに対して，図 6 にオフチップアクセス回数の削減率を，図 7 にプロセッサの性能の向上率を，それぞれ示す．いずれのグラフも横軸はセッティング名であり，より良い結果が得られたものを右側へと記載している．hmean は調和平均である．

図 6 に示すとおり，オフチップアクセスの頻度は平均で 5.4%削減された．最もミスを削減できたセッティングである E-M (Equation Solver Kernel と行列積) では 29.7%の削減率を達成した．こうしたオフチップアクセスの削減を達成しながらも，図 7 に示す性能の向上率については，やはり上述の E-M が 3.2%の性能向上を達成しているが，平均では 0.1%にも満たない変化にとどまった．

性能向上を阻害する要因となっているセッティングは，E および H(Equation Solver Kernel および姫野ベンチマーク) からなるセッティングである．この中でも特徴的なセッティングは E-E であり，全セッティングの中でも 4 番目に高い 9.5%のオフチップアクセス削減を達成しつつも，性能は逆に 2.6%低下し，全セッティングの中でも最悪にまで落ち込んでいる．この原因は，オフチップアクセス削減による利益を打ち消すほど L2 ヒットの平均レイテンシが増大したためであると考えられる．次節でこのような性能変化の原因について更に考察する．

4.3 性能変化の原因に関する考察

図 8，図 9 は，ElasticCC+AS と提案手法のそれぞれについて，一定サイクルごとに区間内のキャッシュパーティショニングの状態，および MPKI(Miss Per Kilo Instruction) の値のログを取り，プロットしたものである．アプリケーションには，前述の 5 種類のベンチマークをそれぞれ異なる順序で並べて連続で実行したものを利用した．このアプリケーションにおけるオフチップアクセスの削減率は 3.9%，性能向上は 0.2%であった．いずれのグラフも，横軸には 1 単位を 100K サイクルとする実行サイクル数を取り，上のグラフの縦軸にはコアに占有のウェイを色付きで示し，下のグラフの縦軸は 4 コアごとにまとめた MPKI 値を示す．

最も特徴的な違いは E(Equation Solver Kernel) にみられる．開始から 100 単位程度のパーティショニング状態のグラフから見られるように，従来手法の ElasticCC+AS では可能な限り共有領域を増加させようとしている一方で，提案手法では可能な限り占有領域を増加させようとしている．図 5 で既に述べたとおり，E はキャッシュのウェイ数を増加させていくと，あるところから急激にヒット率が増加する，周期的にアクセスされるデータが多く含まれたベンチマークである．このとき従来手法では，占有領域のウェイ数が小さいと占有領域の LRU にヒットしないため，占有領域拡大の必要はないと判断される．対して提案手法では，追い出されたラインのうちごく一部を，潜在的なヒット数測定のためのサンプルと

Vol.2011-ARC-196 No.10
2011/7/27

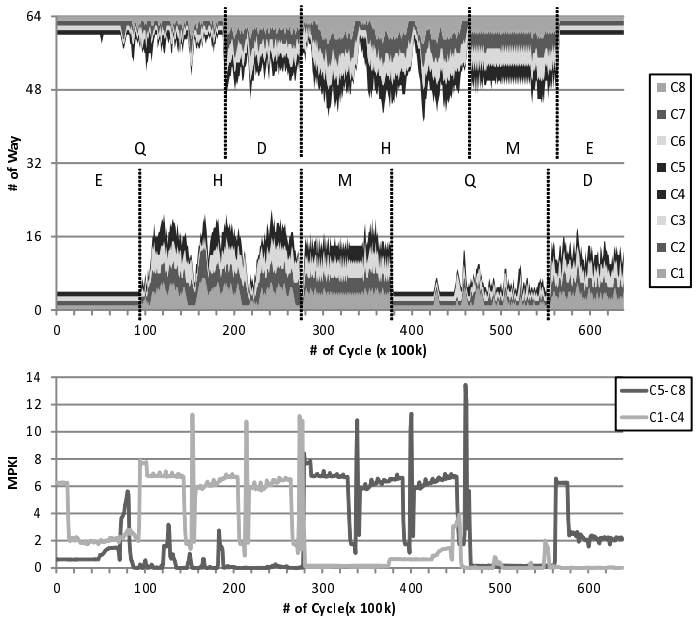

図 8 ElasticCC+AS におけるウェイ配分と MPKI 値.

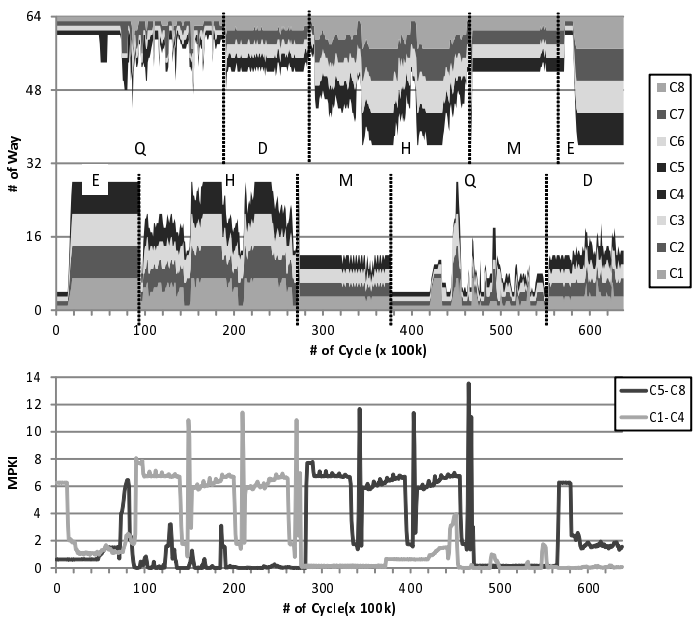

図 9 提案手法におけるウェイ配分と MPKI 値

して他コアに移動しており，そのラインが次の周期でアクセスされた際にヒットする．このヒットを検出して，追い出されたラインを他のコアへと移動することが有効であると判断されれば，占有領域を拡大させて，今度はより積極的に他のコアへのライン移動を行い，ヒット率を向上させる．このように，周期的なアクセスを含むベンチマークに対してデータの再利用を検出可能となったことは提案手法の大きなポイントである．

しかしながら，こうして再利用を検出したことが副作用を生み出したのが，E-E のケースである．このセッティングでは，双方のアプリケーションが占有領域を増加させようとした結果，全てのコアが最大値である 7 ウェイか，あるいはそれに近い占有領域を確保する．これにより共有領域のウェイ数は，最小の場合全コア合計で 8 ウェイにまで縮小される．この状況下で，各コアは追い出されたラインの移動を頻繁に試みる．しかし共有領域のウェイ数はほとんど残されていないため，例えその一部が再利用されてオフチップアクセスを回避したとしても，多くのライン移動が単にオンチップネットワークを混雑させるだけの結果に至り，これが L2 ヒットの平均レイテンシの増大，ひいては性能の低下を招いたと考えられる．

このような性能低下は，各コアが他のコアのキャッシュ占有状況を深く鑑みることなくグリーディに占有領域を拡大してしまったことが要因であると考えられる．従来手法においては，共有領域の LRU ヒットも検出しているため，こうした競争が起こりそうなケースでは，共有領域の LRU ヒットが増加することにより一定の歯止めが得られる．共有領域のウェイ数の合計があまり小さくなりすぎないように，他のコアのキャッシュ占有状況を考慮する形で提案手法を改良することが課題となる．

あるいは，別のアプローチとして，急激にキャッシュヒット率が向上するポイントがわかっているならば，意図的にパーティションに偏りを持たせることで，オフチップアクセスの削減とスループットの向上を図ってもよい．例えば E-E ならば 4 つのコアが 12 ウェイ程度を，残りの 4 つのコアが 4 ウェイ程度を利用するようにパーティションを調整すれば，概ね 35〜40%程度のキャッシュヒット率改善が見込まれる．このようなアプローチは，ソフト

ウェアベースのキャッシュパーティション方式で提案されている[11]．ソフトウェア制御では，コア数の増加による計算量の増加がスケーラビリティを阻害する場合があることには注意が必要であるが，更に柔軟なパーティション制御を求める場合は考慮に値すると考えられる．

以上に提案手法やキャッシュパーティショニングにおける課題点を挙げたが，提案手法においても，E-M のように再利用を検出したことによって非常に良好に働いたセッティングも存在している．こうしたケースを有効に活用しつつ，前述したような課題点を克服していくことで，オフチップアクセスの削減だけではなく，プロセッサ全体の性能を引き上げる可能性は大いに考えられる．

5. おわりに

近年では CMP が主流となっており，CMP のキャッシュを柔軟に利用する方法が提案されている．本稿ではそうした方法のひとつである ElasticCC に対して，RFF を改良した機構を利用したキャッシュパーティショニング方式を提案した．提案手法を利用することにより，オフチップアクセスの頻度は平均 5.4%，最高で 29.7%削減できることを確認し，プロセッサの性能向上を引き出す可能性があること，またそのために解決すべき問題点を明らかにした．

今後は，より規模の大きなアプリケーションにおいても詳細な評価を可能とする態勢を整え，このようなキャッシュ構成において解決すべき点を明らかにし，それを解決するための手法についても順次提案していく．

謝　辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (JST CREST) の「アーキテクチャと形式的検証の協調による超ディペンダブル VLSI」の支援による．

参考文献

1) Kim, C., Burger, D. and Keckler, S.W.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pp.211–222 (2002).
2) Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D. and Keckler, S.W.: A NUCA substrate for flexible CMP cache sharing, *Proceedings of the 19th annual international conference on Supercomputing*, pp.31–40 (2005).
3) Chang, J. and Sohi, G.S.: Cooperative Caching for Chip Multiprocessors, *Proceedings of the 33rd annual international symposium on Computer Architecture*, pp. 264–276 (2006).
4) Herrero, E., González, J. and Canal, R.: Distributed cooperative caching, *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp.134–143 (2008).
5) Beckmann, B.M., Marty, M.R. and Wood, D.A.: ASR: Adaptive Selective Replication for CMP Caches, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.443–454 (2006).
6) Herrero, E., González, J. and Canal, R.: Elastic cooperative caching: an autonomous dynamically adaptive memory hierarchy for chip multiprocessors, *Proceedings of the 37th annual international symposium on Computer architecture*, pp. 419–428 (2010).
7) 小川周吾，平木　敬：メニーコアプロセッサ向き共有キャッシュ配分方式，情報処理学会研究報告 2010-ARC-190，No.2 (2010).
8) 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).
9) 植原　昂，佐藤真平，吉瀬謙二：メニーコアプロセッサの研究・教育を支援する実用的な基盤環境，電子情報通信学会システム開発論文特集号，pp.2042–2057 (2010).
10) Culler, D.E., Gupta, A. and Singh, J.P.: *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann (1999).
11) Chang, J. and Sohi, G.S.: Cooperative cache partitioning for chip multiprocessors, *Proceedings of the 21st annual international conference on Supercomputing*, pp.242–252 (2007).