Vol.2014-ARC-211 No.2
2014/7/28

動的推定によるキャッシュパーティショニング最適化

野村 隼人[1]　力 翠湖[1]　吉見 真聡[1]　吉永 努[1]　入江 英嗣[1]

概要：
本研究では，キャッシュラインの再参照傾向を近似する指標である CC 値を使用し，プログラムの実行フェーズを意識したキャッシュパーティショニング手法について述べる．CC 値の算出をマルチスレッドおよび SDM 向けに拡張する方法について明らかにし，スレッド毎の CC 値の比を元にパーティションを決定する CCP-Rate と，SDM を用いて最適パーティションを探索する CCP-SDM の 2 手法を提案する．両手法をシミュレータに実装し，EQ パーティショニングとの比較を行った．スループット評価では，SDM のモニタリング対象セットへのアクセスの偏りにより調和平均が EQ より低下したものの，最大で 4.1%の向上を得るケースがあり，今後の改良につながる結果となった．

1. はじめに

プロセッサアーキテクチャにおいて，メモリサブシステムの性能は主要な課題の一つである．近年では低次キャッシュ，特に LLC(last level cache) の大容量化が進み，従来のシンプルな局所性利用アルゴリズムを発展させたスマートなキャッシュ制御アルゴリズムの研究が数多く進められている．現在，汎用プロセッサでは，スレッドレベル並列性による高効率なスループット向上を活用するために，ハイエンド用途から組み込み用途に至るまで，マルチコア構成が幅広く採用されている．このような構成では，低次キャッシュを複数のコアが共有するアーキテクチャを持つことが一般的である．例えば Intel Haswell では 8MB の L3 キャッシュを 4 個のコアが共有する [1]．モバイル機器向けプロセッサでも同様に，NVIDIA Tegra3 では 1MB の L2 キャッシュを 4 個のコアが共有している [2]．

共有キャッシュではリソース配分が重要なことが知られている．例えば，頻繁なスキャンアクセスによってキャッシュを溢れさせるようなスレッドが実行されている場合，キャッシュを共有する他のスレッドの有効データが頻繁に追い出されるケースが発生する．スレッド毎に最大ウェイ数を制限するパーティショニング手法の導入によって，特定のスレッドのみ偏ってキャッシュ容量を浪費されることを防ぎ，総合の処理スループットや公平性が向上することが報告されている [3][4][5][6]．

単純な等分ではない，動的な最適パーティショニングではプログラムのフェーズやキャッシュ性能傾向をどのように検出してスレッド割り当てを偏らせるかが課題となる．従来手法ではこの検出にミス率や MPKI のような指標が用いられている．しかし，ミス率や MPKI はプログラムやフェーズによって全く異なる値をとるため，閾値の設定や変更する方向の判断が難しい．例えば，ミス率が高い場合，キャッシュフレンドリー [6] なプログラムであれば割り当てを増やすべきであり，一方でキャッシュインセンシティブなプログラムであれば割り当てを減らすべきである．このように，ミス率そのものは最適なキャッシュ量の指標としては大まかな近似としての効果しか期待できない．

そこで，本論文では CC 値 [7] を用いたキャッシュパーティショニング手法を提案する．CC 値はキャッシュラインの再参照傾向を近似する指標で，容量不足によるスラッシングが発生すると大きく値を落とす性質を持っている．提案手法では SDM[8] を用いて最も CC 値降下の少ない割り当てを探索し，動的にパーティションを変化させる．

提案手法をプロセッサシミュレータ鬼斬弐 [9] rev.6729 上に実装し，スループットおよびパーティション変更の様子を計測した．

以下，2 章では関連研究について述べ，3 章では本研究で提案する手法について述べ，マルチスレッド環境および SDM への CC 値算出手法の拡張を明らかにする．また，CC 値を利用する最適パーティション探索アルゴリズムを明らかにする．5 章では，提案手法を評価・考察し，6 章で結論を述べる．

[1] 電気通信大学大学院情報システム学研究科
Graduate School of Information System, the University of Electro-Communications

表 1 Sanchez らによるキャッシュアクセスの分類

Insensitive	410.bwaves, 465.tonto
Cache-friendly	401.bzip2, 436.cactusADM
Cache-fitting	470.lbm, 483.xalancbmk
Thrashing/Streaming	429.mcf, 433.milc

2. 関連研究

2.1 キャッシュ性能傾向とパーティショニング

Sanchez らの研究 [6] では SPEC CPU 2006 benchmark suite[10] のキャッシュのアクセスパターンを分類している．(表 1)

Insensitive：プログラムの実行に必要なデータがそもそも少なく，ほぼ全てキャッシュに乗りきる場合．最低限のキャッシュ領域があればメインメモリへのアクセスは少ない．

Cache-friendly：再参照されるデータが部分的にキャッシュに乗っている場合．キャッシュ領域が増強できればメインメモリへのアクセスが減らせる．

Cache-fitting：キャッシュがある一定容量を超えた途端にプログラムの実行に必要なデータが乗りきる場合．

Thrashing/Streaming：再参照されるデータ量がキャッシュサイズより大きいか，あるいは一度しかアクセスされない場合．このため，キャッシュ容量を縮小しても拡大しても性能が変わらない．

上に挙げたようなキャッシュ参照傾向は，共有キャッシュの割り当て戦略に関連する．例えば，2 スレッドに共有されるキャッシュにパーティショニングの適用がない (大域 LRU) 構成を仮定する．スレッド 0 で Thrashing パターンの，スレッド 1 で Cache-friendly パターンのプログラムが動作すると，スレッド 0 により次々にデッドブロックがキャッシュに載せられ，代わりにスレッド 1 の有効なラインが追い出されてしまう．このような悪影響は，EQ パーティショニングにより緩和することができる．EQ パーティショニングは，共有キャッシュをウェイ単位で分割し，それぞれのスレッドに等しい数だけ競合時の保持ウェイ上限数を設定する手法である．Cache-friendly なフェーズのプログラムを実行するスレッド 1 に，スレッド 0 に侵食されない領域が与えられることになり，同時に，スラッシングを起こすスレッド 0 は最大でも半分の領域しか浪費しないように制限が掛かることになる．

さらに，この例について考えると，Thrashing パターンのスレッド 0 に割り当てる領域は少なく，Cache-friendly パターンのスレッド 1 に割り当てる領域を多くすることができれば，よりキャッシュを効率的に活用するパーティションとなる．実行中のプログラムフェーズの参照傾向を推定し，プログラムが必要とするキャッシュサイズに合わせたウェイ数の割り当てができれば，不必要なキャッシュ割り当てを減らし，かわりにキャッシュ容量を必要とするスレッドへのリソース割り当てを増やすようなパーティショニングが実現できる．

2.2 ヒット/ミス数を指標とするキャッシュパーティショニングに関する研究

Stone ら [3] は，シングルスレッドで複数プロセスが切り替えられながら利用するキャッシュのミスを抑える事を目的に，キャッシュをウェイ単位で分割する手法を提案した．この研究では，あらかじめキャッシュミスを最小化したい組み合わせのプログラムを実行し，ミス数が最小になるウェイ数の割り当てを貪欲法により求める．この手法は前もって決めたプログラムの組みについて最適化を行い，事前実行において得られた最適なウェイ数の割り当てを以降の実行に適用するものである．しかし，事前実行時と同じ組み合わせのプログラムの実行であっても，プログラムの実行開始タイミングや，プログラムの処理するワーキングセットが事前実行時と異なることで同時期に実行されるフェーズの重ね合わせにずれが生じ，事前実行時に期待したミス数低減が得られない場合が存在する．

Suh ら [4] は，Stone らのシングルスレッド，静的解析のパーティショニングに対して，マルチコアプロセッサ上の共有キャッシュの動的なウェイ分割手法を提案した．この手法では，各スレッドごとに割り当てウェイあたりのミスの減少数 “ marginal gain ” を指標として定義し，この値が最大になるウェイ分割数を探索する．Suh らの提案手法は，この “ marginal gain ” を各コア各 way ごとに調べ，一定のモニタリング期間分記録するために多くのハードウェア資源を必要とする．

Qureshi ら [5] は，Suh らの手法に対して，より少ないハードウェア資源で “ marginal gain ” を観測する手法 (UCP) を提案した．この手法では新たに Utility Monitor(UMON) を追加する．UMON はコア 1 つに 1 つ実装され，“ marginal gain ” の観測のためにウェイごとのキャッシュヒットのカウンタと，ヒット判定のためのタグ情報を持つ．また，キャッシュアクセス数，ヒット数の観測対象を

2.3 Cache Convection

前節で見たように，最適なパーティションを期間あたりのミス率の増減が用いられることが多い．しかし，ミス率はプログラムやフェーズやキャッシュ容量によって大きく変化するため，最適容量の探索のためには Suh らの手法や Qureshi らの手法のように時間やハードウェアコストをかけて，全てのケースを観測しなければならない．

一方，一つのラインが何回アクセスされるかというキャッシュ再参照回数を考えると，この値はどのようなデータ構造を処理しているかに応じて決まった最大値を持つ．もしキャッシュの容量不足からスラッシングが発生する場

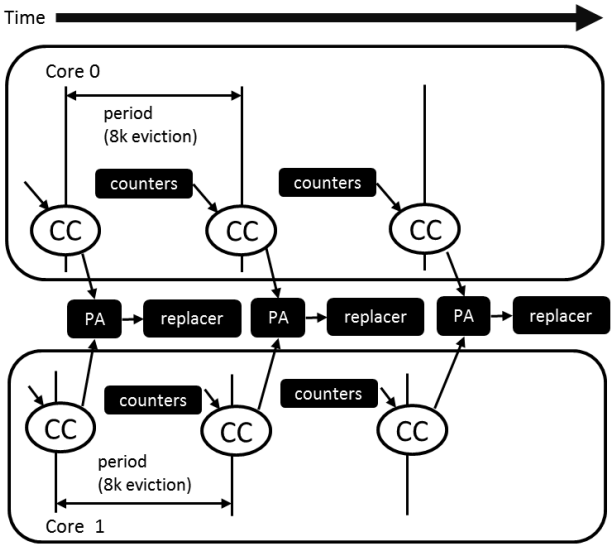

図 1 CC 値の更新タイミング

合，再参照の途中でラインが追い出された後再び挿入されることになるため，ラインあたりの再参照回数は大きく減少する．このような傾向を得る指標として我々は CC 値を提案し，プリフェッチ量の制御に用いている [7]．CC 値はキャッシュラインの再参照傾向を近似する指標であり，以下の式で計算される．

$$newCC = \frac{hitcount}{\frac{accessed\_evicted}{evicted} \times N} \quad (1)$$

ここで，hitcount はキャッシュのヒット数を示しており，キャッシュヒットの度にインクリメントされる．accessed_evicted はキャッシュに乗せた後一度でもアクセスされたラインが追い出された数をカウントしたもの，evicted は追い出しの回数，N はキャッシュエントリ数を示している．

CC 値の計算は evicted が一定値に達するごとに行なわれる (図 1) ノイズの影響を軽減するために前の期間の値が畳み込まれる．

$$accumlatedCC = \frac{previousCC}{2} + \frac{newCC}{2} \quad (2)$$

CC 値の算出後，各カウンタは 0 にリセットされ，次の期間の計測を開始する．

3. CCP

3.1 CC 値のマルチスレッド拡張

本研究では CC 値を割り当て way 数を決定する指標として使用することでスループットの向上を目的とする手法，CCP(Cache Convection-Based Partitioning) を提案する．

CC 値はスレッド毎に計測する性質のものである．まず，本研究では，共有キャッシュを使用する複数のスレッドについてそれぞれの CC 値を計測するための拡張を行う．前章で示したように，CC 値の計測には num_hit, num_evicted_accessed, num_evicted の 3 つのカウンタを用いる．この 3 つについてスレッドの数だけ用意し，以下のように動作を変更する．

num_hit
: キャッシュにヒットしたときに，ヒットしたラインの属するスレッドに対応する num_hit カウンタをインクリメントする．

num_evicted_accessed
: 一度でもアクセスされたことのあるキャッシュラインが追い出されたときに，その追い出されたラインの属するスレッドの num_evicted_accessed カウンタをインクリメントする．

num_evicted
: キャッシュラインが追い出された時，その追い出されたラインの属するスレッドの num_evicted をインクリメントする．あるスレッドの num_evicted が一定数に達した時，そのスレッドの CC 値が更新される．このように，CC 値の更新はスレッドごとに独立したタイミングで行われる．

3.2 CCP-Rate: CC 値を基準としたパーティション決定手法

以降では，スレッドごとに算出された CC 値を用いて最適パーティションを探索する手法を 2 通り提案する．1 番目の手法 CCP-Rate は両スレッドの CC 値の比率に応じてパーティションを決定する手法である．

CC 値はラインの再参照傾向を示すため，CC 値がより高いスレッドの方が何度も参照されるラインを多く含んでいることが予想される．そこで，各スレッドの CC 値の比をとり，次期間のウェイ配分を決定する．その後，全てのスレッドの CC 値の更新が行われるのを待って配分の再決定が繰り返される．CC 値が低い場合でも最低 1 ウェイの割り当ては保証される．

CC 値はデッドラインを多く含むフェーズでは高くなる傾向があるため，単純な比率に基づく配分では高い精度は期待できないが，一方で本アルゴリズムは一回の更新で大きくパーティション配分を変更できる利点がある．

3.3 CCP-SDM: SDM を利用したパーティション決定手法

CC 値は，キャッシュ容量が不足すると急激に減少する傾向があるので，減少量がすくなくなるようにパーティションを変更させていくことで，最適な割り当てを探索することができる．このように CC 値を絶対値ではなく，割当量を変化させたときの増減を利用することでより正確にプログラムがキャッシュを求める傾向を抽出することができる．

増減を観測するためには，異なるパーティションで計測した CC 値を得なければならない．そこで，SDM[8] を用

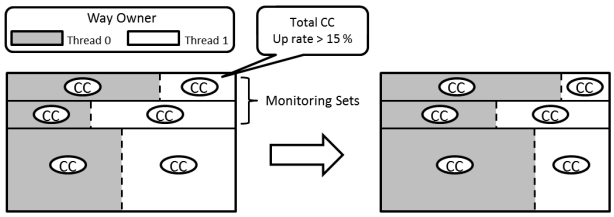

図 2 SDM によるパーティション方針の決定

いて，異なる条件の計測を同時に行う手法を提案する．

モニタセットの CC 値算出は次のように行う．モニタリングセットとして，ウェイ配分を 1 ずつ増減させた 2 種類を用意する．32 ラインに 1 つはスレッド 0 の割り当てを 1 多くして，32 ラインに 1 つはスレッド 1 の割り当てを 1 多くする．スレッド数が多い場合には PTRP のように時分割を併用してモニタを行う方法が考えられるが，本論文では 2 スレッドの場合に限定して議論を行う．モニタリングセットの CC 値を算出するために，num_hit と num_evicted_accessed のカウンタをモニタリングセット用に増設し，それぞれの該当ラインをカウントする．スレッドの num_evicted のカウントに同期して CC 値を計算することにより，算出される 3 つの CC 値は比較が可能な値となる．

全スレッドの CC 値の更新が完了するたびにパーティションの変更を行う．各スレッドについて，現パーティションの CC 値に対する二つのモニタリングセットの CC 値の予測変化量の割合をみて，合計値の大きいパーティションを選択する．

モニタリングセットで調べたパーティションが候補となるので，一度の割り当て変化量は 1 ウェイずつとなるが，更新を繰り返しながら最適なパーティションに近づいていく．(図 2)

4. 実装

4.1 提案手法の概要

5. 評価

5.1 評価環境

CCP-Rate および CCP-SDM を鬼斬 2 rev.6729 上に実装して性能評価を行った．また，比較対象として LRU(パーティショニング適用無し)，EQ パーティショニングモデルを実装した．なお，これらのパーティショニング手法はラストレベルで共有キャッシュである L2 キャッシュにのみ適用しており，L1 キャッシュには LRU を適用している．

評価対象のプログラムとしては SPEC CPU 2006 benchmark suite から表 1 に示したベンチマークの組みを用いた．

SDM でモニタリング対象とするセットへのアクセスの偏りの影響を調べるために，同じ組み合わせでスレッド 0 とスレッド 1 が入れ替わったワークロードについても性能評価をおこなっている．各ワークロードについて，スレッドの先頭 5G ずつをとばしたあとの挙動をシミュレーションして計測した．シミュレーションは，2 スレッドの命令が合計で 1G 命令実行された時点で終了とした．

表 2 ベースラインプロセッサ

Processor	
Core, ISA	2Core 2Thread, Alpha AXP
issue width	int:2, fp:2, mem:2
instruction window	int:32, fp:16, mem:16
branch pred	8KB g-share
BTB	2K entry, 4way
LSQ	64 entry
Cache	
prefetcher	None
L1 I/D Cache	32KB, 4way, 64B line, 3cycle latency
L2 Cache	256KB, 8way, 64B line, 15cycle latency
Memory access	200cycle latency

CCP のパラメタとして，CC 値の更新期間をライン 8k 個の追い出し毎とした．CCP-SDM では CC 値変動の合計 15%以上の変動が見込まれるときにパーティションの変更を行うようにした．また，新しく算出された CC 値が previousCC の 5%以下だった場合にはフェーズ変更検出とした．

5.2 スループット評価

図 3 に 2 スレッド合計の IPC を示す．また, 図 4 に, 大域 LRU(パーティショニング適用無し) のときの合計 IPC を 1 として，相対 IPC で示した．シミュレーション時間の関係で完走しなかったデータは空欄になっている．cactus-lbm や mcf-tonto のように提案手法が EQ を上回る性能を示す組み合わせもあるが，相乗平均では EQ パーティショニングを 1.9%下回る結果となった．提案手法では，tonto-lbm と lbm-tonto のように同じ組み合わせでもスレッド 0 と 1 が変わった時に結果が変わるものがあり，SDM のサンプリングの偏りの影響が要因として挙げられる．

5.3 パーティション変化の様子

提案手法によるパーティション変化の例を図 5 から図 8 に示す．図 5 では cactusADM と lbm を同時に実行したときの CCP-Rate によるパーティションの変化が時系列でプロットされている．この図では cactusADM に多くのラインが割り当てられていることが分かる．このパーティションにより，cactus の実行は加速される一方で lbm の性能低下は大きくなく，図 3 に示されるように総合 IPC の向上につながっている．図 6 は CCP-SDM によるパーティション選択の様子で，同様に cactusADM に多くのラインを割り当てて総合性能を向上させている．

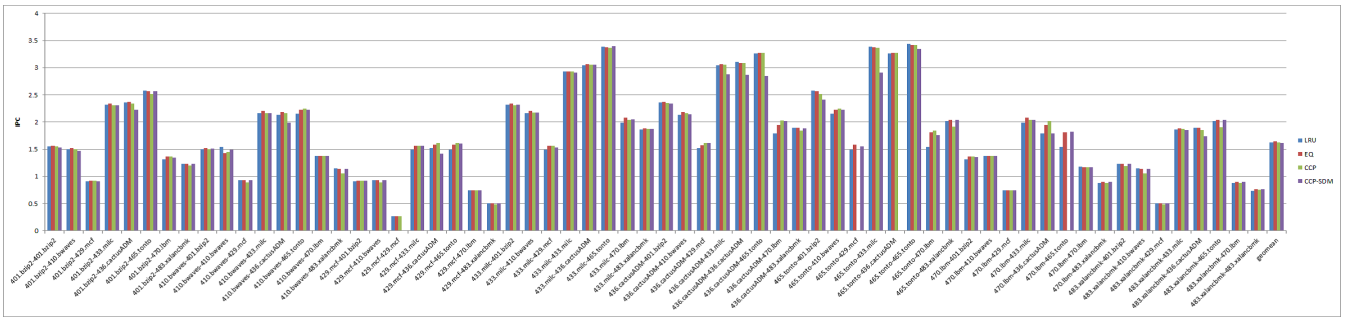

図 3 IPC(2 スレッド分の合計値)

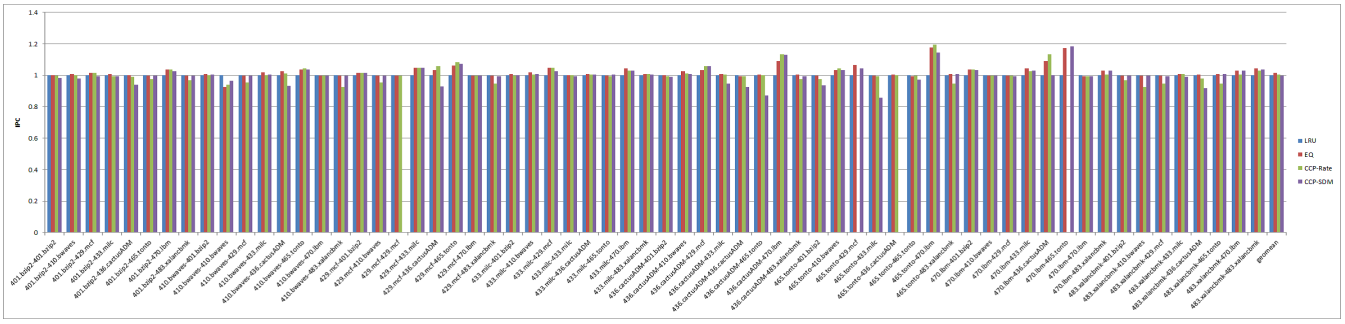

図 4 LRU の IPC を 1 として正規化した IPC の向上率

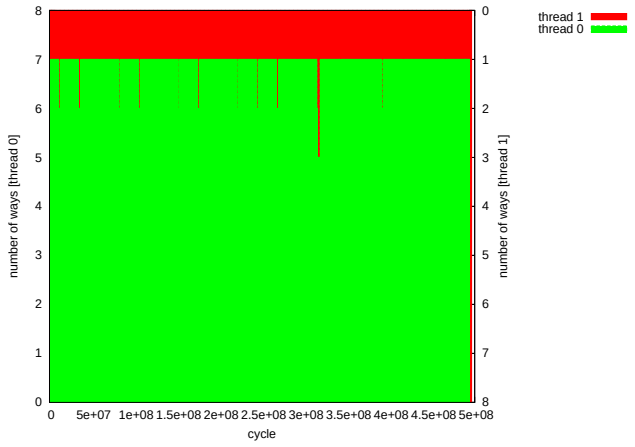

図 5 CCP-Rate でのパーティション履歴 cactusADM[0]-lbm[1]

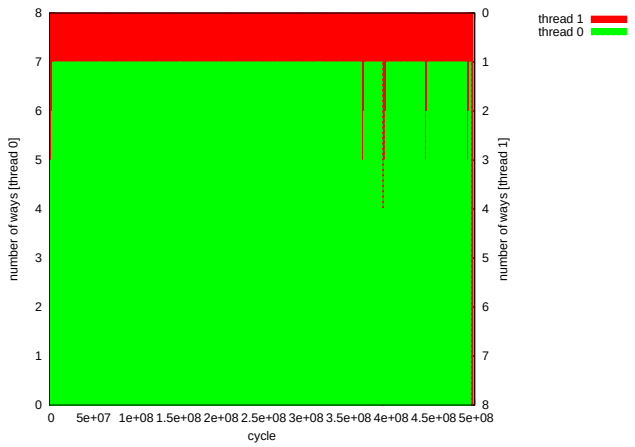

図 6 CCP-SDM でのパーティション履歴 cactusADM[0]-lbm[1]

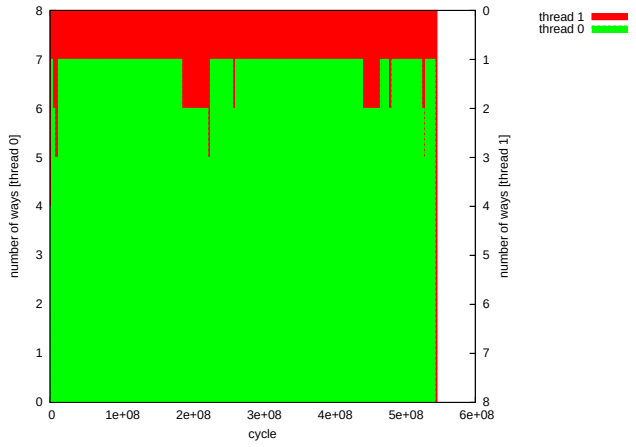

図 7 CCP-Rate でのパーティション履歴 tonto[0]-lbm[1]

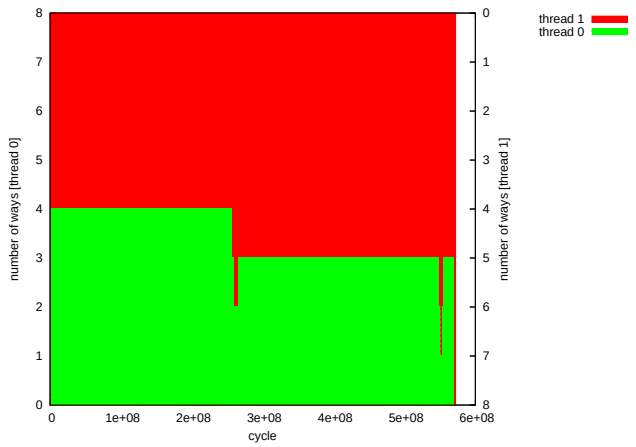

図 8 CCP-SDM でのパーティション履歴 tonto[0]-lbm[1]

一方図 7 と図 8 は tonto-lbm について同様のパーティション変動のタイムラインである．CCP-Rate では tonto に多くのラインを割り当てて実行を加速させたのに対し，CCP-SDM ではやや lbm に多くのラインを割り当て，その結果 EQ よりも性能を落としてしまっている．スレッド番号を入れ替えた lbm-tonto の組み合わせでは異なるパーティションを選択して性能を向上させており，SDM の偏りが性能に影響してしまうことが分かる．

6. おわりに

本論文ではキャッシュパーティショニングに CC 値を用いる手法を提案した．提案手法では CC 値をマルチスレッドおよび SDM に対応させて拡張し，変化をもとに動的にパーティションを変更する．シミュレーションによる評価では，全ベンチマークを通した安定した性能向上は得られなかったが，割り当てを偏らせたほうが良いケースについて，一部の組み合わせでは的確なパーティションを選ぶ動作が見られた．

今後の課題として，モニタリングの改良を通してリソース配分精度を向上させることが挙げられる．

謝辞

本研究の一部は JSPS 科研費 25730028 の助成による．

参考文献

[1] Intel: ARK — Intel Core i7-4770K Processor (8M Cache, up to 3.90 GHz), http://ark.intel.com/products/75123/ (2013).
[2] NVIDIA: Tegra 3 Multi-Core Processors — NVIDIA (2012).
[3] Stone, H. S., Turek, J. and Wolf, J.: Optimal partitioning of cache memory, *IEEE Transactions on Computers*, Vol. 41, No. 9, pp. 1054–1068 (1992).
[4] Suh, G. E., Rudolph, L. and Devadas, S.: Dynamic Partitioning of Shared Cache Memory, *J. Supercomput.*, Vol. 28, No. 1, pp. 7–26 (2004).
[5] Qureshi, M. and Patt, Y.: Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches, *39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006. MICRO-39*, pp. 423–432 (2006).
[6] Sanchez, D. and Kozyrakis, C.: Vantage: Scalable and Efficient Fine-grain Cache Partitioning, *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pp. 57–68 (2011).
[7] Irie, H., Miyoshi, T., Honjo, G., Hiraki, K. and Yoshinaga, T.: Using Cacheline Reuse Characteristics for Prefetcher Throttling, *IEICE TRANSACTIONS on Information and Systems*, Vol. E95-D, No. 12, pp. 2928–2938 (2012).
[8] Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Steely, Jr., S. and Emer, J.: Adaptive Insertion Policies for Managing Shared Caches, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pp. 208–219 (2008).
[9] 塩谷亮太, 五島正裕, 坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装, 先進的計算基盤システムシンポジウム 2009 ポスター (2009).
[10] Standard Performance Evaluation Corporation: SPEC CPU 2006 Benchmark Suite, http://www.spec.org/cpu2006/.