DEIM Forum 2011 E5-4

CC-PAID におけるデータベースサイズと CPU キャッシュ利用効率の関係について

松原　裕貴† 　宮崎　純† 　藤澤　誠† 　天野　敏之† 　加藤　博一†

† 奈良先端科学技術大学院大学 情報科学研究科　〒 630-0192 奈良県生駒市高山町 8916 番地の 5
E-mail: †{yuki-m,miyazaki,fujis,amano,kato}@is.naist.jp

あらまし　本研究では，CC-PAID アルゴリズムにおける評価条件変更時の処理性能の調査を行う．CC-PAID は時系列パターンマイニングアルゴリズム PAID の CPU キャッシュ利用効率に着目し，アクセスパターンの改良による時間的局所性を向上させたものである．PAID 及び CC-PAID を用い，データセットのパラメータ等の評価条件を変更し，処理時間とキャッシュミスの観点から CC-PAID の CPU キャッシュ利用効率の改善効果を明らかにする．加えて，両アルゴリズムによる並列処理時の CPU キャッシュ利用効率等の検証も行う．

キーワード　時系列パターンマイニング，性能評価

Relationship between database size and CPU cache utilization in CC-PAID

Yuki MATSUBARA†, Jun MIYAZAKI†, Makoto FUJISAWA†, Toshiyuki AMANO†, and Hirokazu KATO†

† Graduate School of Information Science，Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630–0192 Japan
E-mail: †{yuki-m,miyazaki,fujis,amano,kato}@is.naist.jp

1. はじめに

時系列パターンマイニングとは，時系列データにより構成されたデータベース (以降 時系列データベース) から，最小支持度と呼ばれる閾値を満たす出現順序を維持した状態の部分列を時系列パターンとして発見する手法であり，データ解析，行動予測，情報推薦といった分野において重要な技術とされている．

一方，時系列パターンマイニングでは大規模なデータベースや最小支持度が小さい場合，多くの処理時間が必要とされることが問題となっており，解決すべき重要な問題として，過去の研究においてアルゴリズムの提案や並列化による処理時間の改善が取り組まれてきた．しかしながら，情報爆発時代の今日においては処理対象となる多くのデータベースはより大規模になると想定され，それに伴い処理時間も大幅に増加することが考えられる．このため，時系列パターンマイニングの処理時間の改善は今後も重要な課題の一つと考えられる．

近年では，これらの問題に対してアルゴリズムの提案や並列化といったアプローチ以外に CPU キャッシュの利用効率の改善による処理速度の向上が試みられている [1]．時系列パターンマイニングでは，アルゴリズムの特性上，特定のデータ構造に対して再帰的なアクセスが発生する傾向があり，CPU のキャッシュサイズを超えるようなデータ構造へのアクセスが生じる場合，キャッシュミスによるデータアクセスのレイテンシが発生する可能性がある．CPU のマルチコア化が進む今日においては，CPU の速度向上に対するメインメモリのアクセス速度の差が埋まることは考えられず，メモリの壁 [2] の問題は未だ解決していない．このため，キャッシュミスにより生じるアクセスレイテンシは時系列パターンマイニングの処理においてもボトルネックとなりえ，キャッシュミスの軽減に着目した CPU キャッシュの利用効率の改善による処理時間の短縮は重要な課題である．

そこで，我々は CC-PAID の提案を行った [3]．CC-PAID は，既存の時系列パターンマイニングアルゴリズム PAID [4] に対し，キャッシュ指向メモリアクセスと並列化の二つの高速化手法による改良を行ったものである．本稿では，データセットのパラメータと最小支持度の変更による PAID 及び CC-PAID の処理時間，キャッシュミスカウントに関する評価実験を行い，CC-PAID の有効性を明らかにする．また，両アルゴリズムで

並列処理を行った際の処理時間とキャッシュミスカウントの計測も行う．

2. 関連研究

Yang らは時系列データに含まれるアイテムの最終出現位置のリスト (以降 ILP-list) 上で，長さ k の時系列パターンより後に出現しないアイテムを調べることにより，長さ $k-1$ の時系列パターンの処理で得たアイテムの出現回数から出現しないアイテムを差し引いていくことにより，最小支持度を満たすものを調べ，長さ $k+1$ の時系列パターンの発見を行う PAID アルゴリズムの提案を行っている [4]．長さ k の時系列パターンより後に出現しないアイテムを調べる方法としては，時系列データ上での長さ $k-1$, k の時系列パターンの位置 (以降 $k-1$, k-pbp) を ILP-list に合わせ，$k-1$-pbp より後, k-pbp までの範囲内のアイテムを非出現とする．これにより，処理するデータ範囲が縮小されるため，他のアルゴリズムよりも効率的に処理が可能とされている．

また，Ghoting らは相関ルールマイニングのアルゴリズムのデータ構造に対し，連続したアクセスができるようなデータの再配置やデータサイズを抑えるといった改良，CPU キャッシュに収まるようにデータを分割してフェッチしたデータを無駄なく利用する等の改良を空間的局所性，時間的局所性の観点から行うことにより CPU キャッシュを有効的に利用した速度向上を達成している [1]．

PAID アルゴリズムの ILP-list は出現しないアイテムの探索と出現回数からの差し引きのための処理で非常に多くの再帰的なアクセスが生じるデータ構造であり，時系列データベースの規模や最小支持度によっては多くのキャッシュミスが生じる可能性がある．CC-PAID では，主に ILP-list で生じるキャッシュミスについて着目し，アクセスパターンの改善による時間的局所性の向上を行い，キャッシュミスの軽減を可能とする．

3. CC-PAID

3.1 Common-Prefix-At-A-Time

PAID では時系列パターンを表わす辞書式順序木 [5] 上で，深さ優先探索により単数の長さ k の時系列パターンを選択し，長さ $k+1$ の時系列パターンの発見処理を決定する．ILP-list の処理は対応する時系列データの ID(以降 SID) に対して昇順に行われていくため，CPU キャッシュよりも遥かに大きいサイズの時系列データベースの処理を行うと，一つ前の長さ $k+1$ の時系列パターンの発見処理が終了した際に SID の値が大きい ILP-list しか CPU キャッシュに残っていなく，次の長さ $k+1$ の時系列パターンの発見処理開で再び SID の小さな ILP-list をフェッチしなくてはならない可能性が高くなり，キャッシュミスの原因となる．一度アクセスされた ILP-list が直ぐに再アクセスされないため，時間的局所性が良くないと言える．例として，ある時系列データベースに対して処理を行い時系列パターン A → B を処理するまでの PAID のアクセスパターンを示す (図 1)．なお枠で囲まれた時系列パターンは処理が終了していることを示し，破線は処理対象となっていることを示す．この

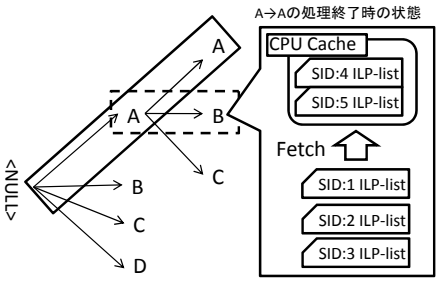

図 1　PAID のアクセスパターンと ILP-list のキャッシュミス

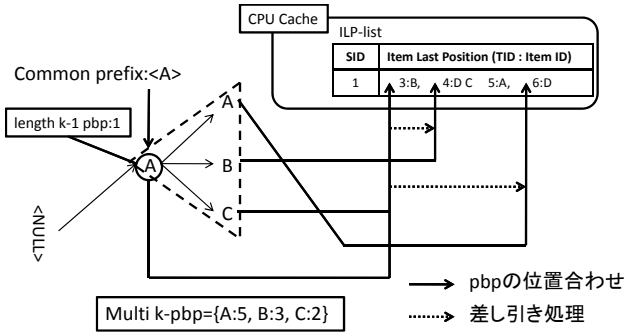

図 2　CC-PAID のアクセスパターンと Common-Prefix-At-A-Time による ILP-list の処理

例では時系列パターン A → A の処理終了時に CPU キャッシュに SID 4，5 の ILP-list しか存在していない．このため，時系列パターン A → B の処理開始時に SID 1，2，3 をフェッチする必要が生じる．

ここで，ILP-list へのアクセスを考察すると，共通する接頭辞（以降 *common prefix*）を含む長さ k の時系列パターンが複数存在する場合，非出現となるアイテムを探す処理の対象範囲は同じとなる．これは $k-1$-pbp によって ILP-list が処理対象となるかを判断することができ，非出現となるアイテムを調べる際，ILP-list 上で $k-1$-pbp に対応する位置より後が非出現となるアイテムを探索するための範囲となるためである．*common prefix* を含む時系列パターンの例として，図 1 の長さ 2 の時系列パターンでは A → A，A → B，A → C が *common prefix*: A を含む時系列パターンである．

Common-Prefix-At-A-Time では各々の SID の処理で *common prefix* を含む複数の長さ k の時系列パターンを一括して処理することにより，時間的局所性を向上させる．図 2 は SID 1 の ILP-list を *common prefix*: A で処理した例である．始めに $k-1$-pbp（*common prefix*: A）を用いて，時系列パターン A → A，A → B，A → C のそれぞれの位置を k-pbp として調べる．得られた複数の k-pbp を，SID の単位で Multiple k-pbp (以降 multi-k-pbp) としてまとめる．次に，SID 1 の ILP-list を対象に時系列パターン A → A の k-pbp の位置合わせを行い，非出現となるアイテムを一つ前の処理 (時系列パターン A) で得られたアイテムの出現回数から差し引くことにより時系列パターン A → A 以降の範囲のアイテムの出現回数の更新を行う．続けて A → B，A → C で順に k-pbp の

表 1 実験環境

OS	Fedora 13 64bit	
カーネル	2.6.34.7-61	
CPU	Intel Core i7 980X	
	周波数	3.33GHz
	コア数	6
	L2 cache size	256KB × 6
	L3 cache size	12MB
メインメモリ	12GB	

位置合わせと非出現となるアイテムの差し引き処理を行い，それぞれの時系列パターン以降の範囲のアイテムの出現回数を更新する．SID 1 に関する時系列パターン A → A，A → B，A → C の処理が全て終了すると SID 2 以降の処理もそれぞれ同様に行う．全 SID の処理が終了した際に時系列パターン A → A，A → B，A → C のそれぞれの投影データベース内のアイテムの出現回数が決定される．Common-Prefix-At-A-Time により時系列パターン A → A の処理でアクセスされた ILP-list が，時系列パターン A → B，時系列パターン A → C の処理ですぐにアクセスされるため，CPU キャッシュに存在するデータの再利用率が向上し，時間的局所性の向上により，キャッシュミスを軽減することが可能となる．

3.2 並 列 化

PAID アルゴリズムで並列化可能な処理は，主に SID 毎の $k-1$，k-pbp の取得，ILP-list 上での位置合わせと差し引き処理であり，CPU コア数に応じて時系列データの単位で分割可能である．また，時系列パターンマイニングでは時系列データ数が非常に多い場合が考えられるため，データ分割を採用した．CC-PAID では，並列処理により，それぞれの SID 毎の Common-Prefix-At-A-Time による処理を行う．

4. 評 価 実 験

本節では，時系列パターンマイニングアルゴリズム PAID と CC-PAID を評価対象とし，特徴の違うデータセットにより，処理時間，キャッシュミスの評価を行う．また，並列処理時の処理時間及びキャッシュミスについての評価も行う．なお，PAID 及び CC-PAID は S-Step [5] のみの評価となる．

実験環境を表 1 に示す．プログラムのコンパイルには g++ (GCC，version 4.4.5) を用い，最適化オプションとして-O2 を指定し，プログラムの生成を行った．キャッシュミスの計測には Intel VTune Amplifier XE 2011 を用い，L2, L3 キャッシュミスの見積もりを計測した．次に，実験に用いるデータセットを示す．IBM Data Generator により 5 つデータセットの生成を行った．一つは時系列データ数 50000，時系列データ内の平均トランザクション数 50，トランザクション内の平均アイテム数 5，アイテムの種類数 400 の N50S50T5NI04 であり，残りの 4 つは N50S50T5NI04 の時系列データ数を 2 倍の 100000 と設定した N100S50T5NI04，N50S50T5NI04 の時系列データ内の平均トランザクション数を 2 倍の 100 とした N50S100T5NI04，N50S50T5NI04 のアイテムの種類数を 2 倍の 800 にした N50S50T5NI08，N50S50T5NI04 の時系列データ数と平均トランザクション数を 2 倍にした N100S100T5NI04 である．データセット N50S50T5NI04 を基準として各パラメータを変更したデータセットを用い，処理時間とキャッシュミスを計測し，CC-PAID の有効性を調べた．また，データセット N100S100T5NI04 は並列処理時の性能評価とキャッシュミスの計測に用いられる．

4.1 最小支持度変更時の処理時間とキャッシュミスの計測

スレッド数を 1 と設定し，データセット N50S50T5NI04, N100S50T5NI04, N50S100T5NI04, N50S50T5NI08 を用い，それぞれで最小支持度を変更した時の処理時間と L2,L3 のキャッシュミスの見積もりを計測した．加えて，それぞれのデータセットで用いた最も小さい最小支持度の時のメモリ使用量の計測も行った．

データセット N50S50T5NI04 では最小支持度を 30〜45%, N100S50T5NI04 では 35〜50%，N50S50T5NI08 では 15〜30%と変化させた．データセット N50S100T5NI04 では処理時間が大幅に増加したため，最小支持度を 65〜80%と設定した．

4.1.1 処理時間の計測

N50S50T5NI04 を計測した結果 (図 3)，最小支持度を 30%と設定した時，PAID の処理時間が約 630 秒となったのに対し，CC-PAID では約 317 秒となり最大で約 1.99 倍の処理速度の向上が確認できた．

N100S50T5NI04 を計測した結果 (図 4) では，最小支持度を 35%と設定した時，PAID の処理時間が約 500 秒となったのに対し，CC-PAID では約 255 秒となり最大で約 1.95 倍の処理速度の向上が確認できた．

N50S100T5NI04 を計測した結果 (図 5) では，最小支持度を 65%と設定した時，PAID の処理時間が約 1639 秒となったのに対し，CC-PAID では約 830 秒となり最大で約 1.97 倍の処理速度の向上が確認できた．

N50S50T5NI08 を計測した結果 (図 6) では，最小支持度を 15%と設定した時，PAID の処理時間が約 546 秒となったのに対し，CC-PAID では約 304 秒となり最大で約 1.8 倍の処理速度の向上が確認できた．

4.1.2 処理時間の考察

基準とするデータセット N50S50T5NI04 に対して最小支持度を同じ 35%とした時の時系列データ数を 2 倍とした N100S50T5NI04 の CC-PAID の処理時間を見ると，N50S50T5NI04 では約 126 秒，N100S50T5NI04 では約 255 秒となり，約 2 倍の処理時間を要した．これに対し，平均トランザクション数を 2 倍とした N50S100T5NI04 では最小支持度が 65%の時点で約 830 秒となるため，時系列データ数を増加させるよりも平均トランザクション数を増加させた方が処理時間に影響があることがわかる．これは，平均トランザクション数の増加により出力される時系列パターン数の増加とパターン毎の処理が増える可能性が高くなるためと考えられる．

次に，各データセットの計測時の最も大きい最小支持度での PAID に対する CC-PAID の高速化比を調べると N50S50T5NI04 では最小支持度 45%の時，1.77 倍，

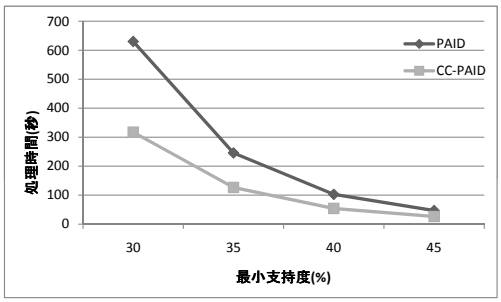

図 3　処理時間の計測：N50S50T5NI04

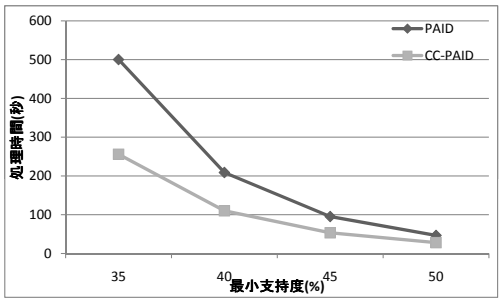

図 4　処理時間の計測：N100S50T5NI04

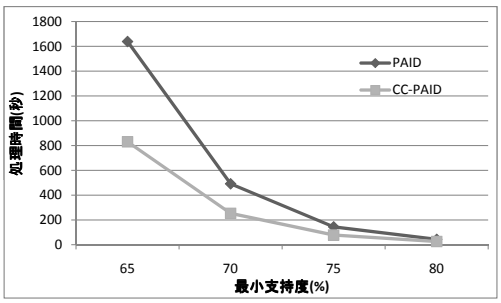

図 5　処理時間の計測：N50S100T5NI04

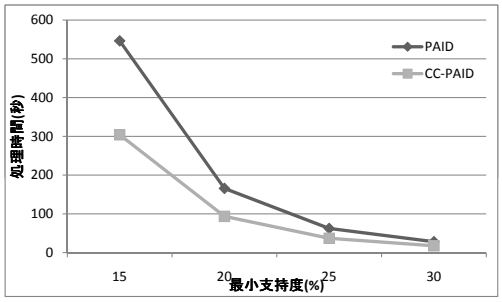

図 6　処理時間の計測：N50S50T5NI08

N100S50T5NI04 では 50%の時，1.66 倍，N50S100T5NI04 では 80%の時，1.67 倍，N50S50T5NI08 では 30%の時，1.61 倍となり，いずれの場合も最小支持度を下げていくほど PAID に対する CC-PAID の高速化が行われることが分かった．

4.1.3　キャッシュミスの計測

N50S50T5NI04 を計測した結果 (図 7, 8)，最小支持度を 30%と設定した時，PAID のキャッシュミスに対し，CC-PAID では最大で L2：約 76%, L3：約 61%のキャッシュミスの削減が確認できた．このとき，PAID に対しての CC-PAID のメモリ使用量が約 38%増加したことが確認された．

N100S50T5NI04 を計測した結果 (図 9, 10) では，最小支持度を 35%と設定した時，PAID のキャッシュミスに対し，CC-PAID では最大で L2：約 75%, L3：約 60%のキャッシュミスの削減が確認できた．このとき，PAID に対しての CC-PAID のメモリ使用量が約 33%増加したことが確認された．

N50S100T5NI04 を計測した結果 (図 11, 12) では，最小支持度を 65%と設定した時，PAID のキャッシュミスに対し，CC-PAID では最大で L2：約 71%, L3：約 58%のキャッシュミスの削減が確認できた．このとき，PAID に対しての CC-PAID のメモリ使用量が約 30%増加したことが確認された．

N50S50T5NI08 を計測した結果 (図 13, 14) では，最小支持度を 20%と設定した時，PAID のキャッシュミスに対し，CC-PAID では最大で L2 で約 11%のキャッシュミスの増加が確認され, 最小支持度を 15%と設定した時，L3 で約 61%のキャッシュミスの削減が確認できた．なお，最小支持度を 15%と設定した時，PAID に対しての CC-PAID のメモリ使用量が約 42%増加したことが確認された．

4.1.4　キャッシュミスの考察

データセット N50S50T5NI08 を除いた全てのデータセットにおいて L2, L3 共にほとんどの場合で 5 割以上のキャッシュミスの削減が確認された．このため，時系列データ数及び時系列データ内の平均トランザクション数に関してのパラメータが変更された場合でも Common-Prefix-At-A-Time では効率的にキャッシュミスを削減できると考えられる．アイテムの種類数を増加したデータセット N50S50T5NI08 では L2 のキャッシュミスが PAID よりも増加したことが確認された．これは，実装においてアイテムの種類数に影響するデータ構造に対し，Common-Prefix-At-A-Time によるアクセスが発生したことが原因となった可能性が考えられるが，L3 のキャッシュミスに関しては削減が確認されており，処理速度も向上している．このため，L3 のキャッシュミスによるレイテンシが L2 よりも大きいことも考慮すると，L3 のキャッシュミスの削減が処理速度の向上に大きく関係していると考えられる．更に，各データセットで用いた最も大きい最小支持度の時の L3 キャッシュミスの削減割合を確認すると，N50S50T5NI04 では最小支持度 45%の時，約 56%，N100S50T5NI04 では最小支持度 50%の時，約 55%，N50S100T5NI04 では最小支持度 80%の時，約 49%,N50S50T5NI08 では最小支持度 30%の時，約 56%の削減となっており，処理速度の計測時と同様に最小支持度を小さくするに従い PAID に対する CC-PAID でのキャッシュミスの削減割合が増加する傾向がある．これは，最小支持度を下げることにより出力される時系列パターン数が増加し，PAID において多くのキャッシュミスが発生するためであると考えられる．処理速度の向上比と L3 キャッシュミスの削減割合から，CC-PAID では最小支持度が小さくなるにつれてキャッシュミスの削減割合が増え，同様の傾向で性能向上が行われるため，Common-Prefix-At-A-Time によるキャッシュミスの削減が処理時間の短縮に貢献できていると考えられる．

また，計測した全てのデータセットにおいて PAID に対し，CC-PAID のメモリ使用量が増加しているが，これは Common-Prefix-At-A-Time により複数の時系列パターンを処理対象と

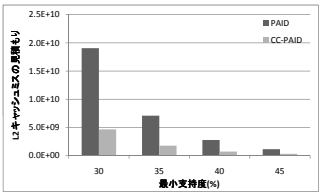

図 7　N50S50T5NI04:
L2 キャッシュミスの見積もり

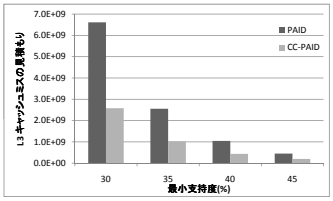

図 8　N50S50T5NI04:
L3 キャッシュミスの見積もり

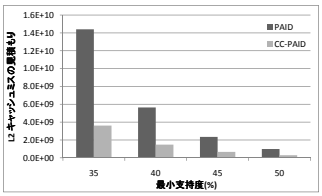

図 9　N100S50T5NI04:
L2 キャッシュミスの見積もり

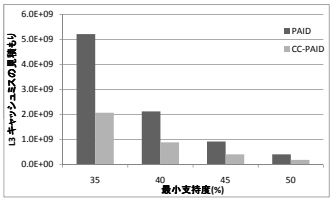

図 10　N100S50T5NI04:
L3 キャッシュミスの見積もり

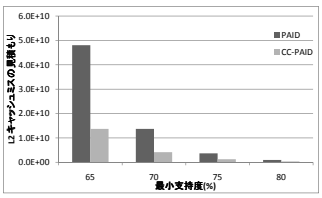

図 11　N50S100T5NI04:
L2 キャッシュミスの見積もり

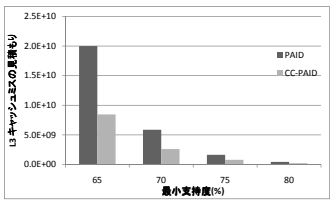

図 12　N50S100T5NI04:
L3 キャッシュミスの見積もり

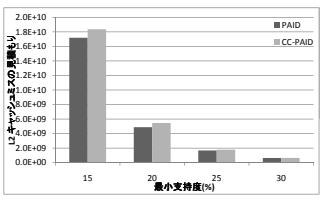

図 13　N50S50T5NI08:
L2 キャッシュミスの見積もり

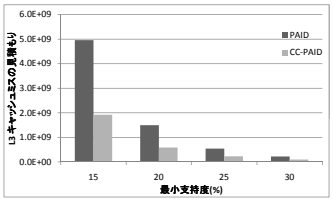

図 14　N50S50T5NI08:
L3 キャッシュミスの見積もり

するため，処理対象の時系列パターン毎のデータを保持する必要があるためと考えられる．

4.2　並列処理による処理時間とキャッシュミスの計測

最小支持度を 70%と設定し，データセット N100S100T5NI04 を用い，スレッド数を 1, 2, 4, 6 と変更した時の処理時間の計測を行った．キャッシュミスに関してはスレッド数を 6 とした時の L2, L3 のキャッシュミスの見積もりの結果を示す．

4.2.1　スレッド数変更時の処理時間

計測結果を図 15 に示す．スレッド数 6 の時に PAID の処理時間が約 233 秒となったのに対し，CC-PAID では約 119 秒となり，最大で約 1.96 倍の処理速度の高速化が確認できた．

スケーラビリティについては，PAID ではスレッド数 1 に対し，スレッド数 6 の時，約 4.3 倍の高速化となり，CC-PAID でもスレッド数 1 に対し，スレッド数 6 の時，約 4.3 倍の高速化となった．

4.2.2　並列処理時のキャッシュミスの計測

計測結果を図 16 に示す．L2, L3 のキャッシュミスの見積もりは各 CPU コアで生じたキャッシュミスの見積もりの合計であり，スレッド数 6 の時に，PAID に対して CC-PAID では，L2：70%, L3：56%程度のキャッシュミスの削減が確認できた．また，他のスレッド数の時もほぼ同じキャッシュミスの見積もりが確認された．

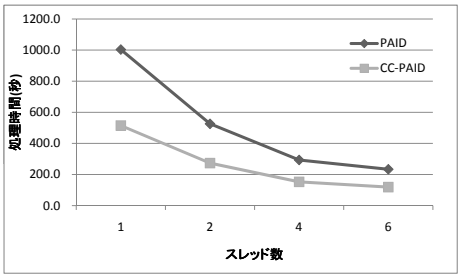

図 15　スレッド数変更時の処理時間の計測

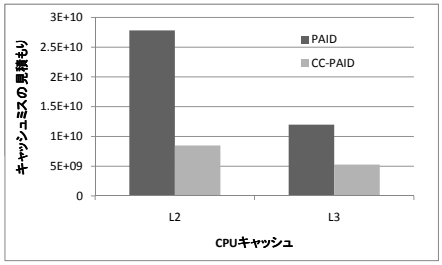

図 16　スレッド数 6 の時の L2, L3 キャッシュミスの見積もり

4.2.3　並列処理に関する考察

並列処理時においても，速度向上とキャッシュミスの削減が確認されたため，シングルスレッド時同様にキャッシュミスの削減により，効果的に処理速度の向上が行われると考えられる．

5.　まとめと今後の課題

本稿では，PAID アルゴリズムの CPU キャッシュ利用効率の改良を行った CC-PAID の処理時間とキャッシュミスの計測を行い，パラメータの違うデータセットを用いた際の有効性の確認を行った．結果として，計測に用いた全てのデータセットで PAID に対しての CC-PAID の処理速度の向上が確認された．キャッシュミスに関しては，アイテムの種類数を増加させた場合を除いて，L2, L3 のキャッシュミスを削減できることが確認できた．更に，CC-PAID では並列処理を行った際もキャッシュミスの削減と処理速度の向上が行われ，有効性が確認できた．

今後の課題としては，Cell Broadband Engine や GPGPU といった他のアーキテクチャを有効的に利用した処理時間の短縮方法の提案が挙げられる．また，アイテムの種類数を増加させた場合に CC-PAID では L2 キャッシュミスが増加することが確認され，よりアイテムの種類数を増やした場合，最もレイテンシの大きい L3 キャッシュにも影響する可能性も考えられるため，アイテムの種類数を増加させた場合のキャッシュミスのより詳細な増加原因の考察と改良も行っていく必要があると考えられる．

謝　　辞

本研究の一部は、科研費補助金基盤研究 (A)(課題番号:

22240005) ならびに若手研究 (B)(課題番号: 21700111) の支援による。ここに記して謝意を表す。

文　　献

[1] Ghoting, A., Buehrer, G., Parthasarathy, S., Kim, D., Nguyen, A., Chen, Y.-K. and Dubey, P.: Cache-conscious frequent pattern mining on a modern processor, Proceedings of the 31st international conference on Very large data bases, VLDB '05, VLDB Endowment, pp.577-588 (2005).

[2] Wulf, W.A. and McKee, S.A.: Hitting the memory wall: implications of the obvi-ous, SIGARCH Comput. Archit. News, Vol.23, pp.20-24 (1995).

[3] 松原 裕貴, 宮崎 純, 加藤 博一 : CPU キャッシュを有効利用した並列時系列パターンマイニングアルゴリズム Cache-conscious parallel PAID の提案, 第 151 回 データベースシステム研究発表会, 2010-DBS-151, pp.1-7(2010).

[4] Yang, Z., Kitsuregawa, M. and Wang, Y.: PAID: Mining Sequential Patterns by Passed Item Deduction in Large Databases, Proceedings of the 10th International Database Engineering and Applications Symposium, Washington, DC, USA, IEEE Computer Society, pp.113-120 (2006).

[5] Ayres, J., Flannick, J., Gehrke, J. and Yiu, T.: Sequential PAttern mining using a bitmap representation, Proceedings of the eighth ACM SIGKDD international con- ference on Knowledge discovery and data mining, KDD '02, New York, NY, USA, ACM, pp.429-435 (2002).