情報処理学会論文誌　データベース　Vol. 4　No. 2　88–100 (July 2011)

CC-PAID：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

松原裕貴[†1]　宮崎純[†1]　藤澤誠[†2]
天野敏之[†3]　加藤博一[†1]

本論文では，既存の時系列パターンマイニングアルゴリズム PAID を対象に，CPU のキャッシュミスの軽減による処理時間の短縮を目的とした改良手法の提案を行う．時系列パターンマイニングでは，データベースの規模やパターンの抽出に用いられる閾値によって非常に長い処理時間が必要とされることが問題とされており，過去の研究においてアルゴリズムや並列化の提案が行われてきた．また，処理対象となるデータベース全体に対して多くの反復的なアクセスが生じ，キャッシュミスが発生しやすいため，大規模なデータベースや小さな閾値を利用した場合，そのレイテンシは全体の処理時間に対して無視できない可能性がある．PAID アルゴリズムにおいても同様に反復的なアクセスが生じるデータ構造があり，キャッシュミスが起こる可能性が考えられる．そのため，提案手法では処理対象へのアクセスパターンの改良により，PAID アルゴリズムの特定のデータ構造での時間的局所性を向上させ，キャッシュミスの軽減を行う．また，処理時間を短縮するために並列化も行い，性能評価によりこれらの有効性の確認を行う．

CC-PAID: A Cache-conscious Parallel Sequential Pattern Mining Algorithm

Yuki Matsubara,[†1] Jun Miyazaki,[†1]
Makoto Fujisawa,[†2] Toshiyuki Amano[†3]
and Hirokazu Kato[†1]

In this paper, we propose a technique to reduce the memory access latency caused by cache misses in PAID algorithm. Since sequential pattern mining algorithms generally require long time depending on the database size and the minimum support value, many methods have been proposed such as new efficient algorithms and their parallelization. The sequential pattern mining tends to repeat database scans, which frequently cause cache misses. The frequent cache misses cannot be ignored, when, in particular, a low minimum support value is set. In PAID algorithm, cache misses happen in some data structures which are accessed repeatedly. Therefore, we propose a method to improve the data structures and their access patterns so that temporal locality of reference increases, which leads to more cache hits. In addition, we also parallelize our method to reduce further execution time. We evaluate the efficiency of our method through some experiments.

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から時系列パターンと呼ばれる出現順序を維持した状態の特定の閾値を満たす部分列を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．

代表的な例としては，マーケティングへの応用があげられる．ある百貨店の 1 年間の購買履歴のデータベースを対象に時系列パターンマイニングを行い，得られた時系列パターンの中から，今まで気がつかれなかった有用な時系列パターンとして，たとえば「ワイングラスを購入した人の 6 割が 3 週間以内にジャケットを購入する」といったことが分かったとする．この結果をもとに，顧客の行動を予測した商品の品揃や価格等の調整といったマーケティング戦略が可能となる．このほかにも Web，生物学的なデータ等の様々な分野での利用が可能であり，処理対象とするデータベースの規模も様々である．

一般に，時系列パターンマイニングにおける処理コストの増加要因として最小支持度とデータベースの規模があげられる．最小支持度とは，時系列パターンの抽出基準として用いられる閾値であり，この閾値が小さいほど多くのパターンの抽出処理が発生する．また，パターンの発見処理では多くのアルゴリズムにおいて，データベース全体に対するスキャンとデータのカウント処理が行われており，これらが処理時間の大半を占めるとされている．このため，小さな最小支持度と大規模なデータベースでは処理時間が膨大になるといった傾向があり，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[1)–7)]，並列化による処理時間の改善[8),9)] が取り組まれてきた．

†1 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†2 筑波大学大学院図書館情報メディア研究科
Graduate School of Library, Information and Media Studies, University of Tsukuba
†3 山形大学工学部システム創成工学科
Department of Systems Innovation Engineering, Yamagata University

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，それにともな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，実際の利用において処理時間に制約がある場合は時系列パターンマイニングの処理を適用できない可能性もある．また，処理時間を減らすために最小支持度を大きく設定することは効果的だが，この場合は有用とされる時系列パターンをとりこぼすといった問題も生じる．さらに，抽出されたパターンを利用して他の処理を行うといったケースも考えられ，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 1 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[10]．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[11] の問題はいまだ解決していない．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うるため，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 の提案と評価を行う．CC-PAID は 2 つの時系列パターンマイニングの高速化手法からなる．1 つはキャッシュ指向メモリアクセスによる高速化であり，もう 1 つは並列化による高速化である．特に，キャッシュ指向メモリアクセスではアクセスパターンを改良することにより，キャッシュミスが発生する可能性のあるデータ構造に対し，それを利用する可能性のある複数の時系列パターンの処理過程で一括したアクセスを行うことにより，時間的局所性の改善を行う．

2. 関 連 研 究

時系列パターンマイニングでは，過去の研究において処理時間の短縮のためにアルゴリズムの開発，改良，または並列処理等の応用を行い，処理時間の短縮に取り組んできている．

Srikant らは，長さ $k-1$ の時系列パターンどうしで長さ k の時系列パターンとなる可能性のある候補シーケンスを結合により生成し，データベース上での候補シーケンスの出現数を調べ，最小支持度と呼ばれるユーザが設定した閾値を満たすものを長さ k の時系列パターンとして発見する GSP アルゴリズムの提案を行っている[1]．

Zaki は，他のアルゴリズムがデータベース全体に対して繰り返しスキャンする傾向があり，そのコストが大きいものとし，長さ $k-1$ の時系列パターンが出現する時系列データの ID と出現した時間を要素とするリスト（id-list）を用い，時系列パターンの組合せを表現するラティス構造上で，長さ $k-1$ の時系列パターンの id-list 間の結合を行い，結合した結果から最小支持度を満たすものを長さ k の時系列パターンとして発見する SPADE アルゴリズムの提案を行っている[2]．また，大規模なデータベースや多くの時系列パターンの処理を行う場合では I/O コストを小さくすることが重要と考え，共通する接頭辞を持つ時系列パターンをクラスとしてまとめ，メインメモリ内で独立してパターンの発見処理を行えるようにクラスごとにラティス構造を分割していく手法の提案も行われている．さらに，Zaki は SPADE アルゴリズムを基に，分散共有メモリマシンによる並列処理を可能とする pSPADE アルゴリズムの提案も行っている．pSPADE では id-list を水平または垂直に分割し，プロセッサごとでデータ並列，あるいはタスク並列で処理する提案を行っている[8]．

GSP のように候補シーケンスを生成し，データベース上で繰り返しスキャンを行うようなアルゴリズムでは，候補シーケンス数が膨大になった場合に多くの処理コストが発生する可能性がある．これに対し，Pei らはデータベース上で直接最小支持度を満たすアイテムを発見し，長さ k の時系列パターンに連結させることにより長さ $k+1$ の時系列パターンを発見する PrefixSpan アルゴリズムの提案を行っている[4]．処理対象の時系列パターンより後に出現するデータ範囲を投影データベースと呼び，投影データベース内で最小支持度を満たすアイテムを発見し，長さ k の時系列パターンの後ろに追加して，長さ $k+1$ の時系列パターンの発見を行う．時系列パターン長が長くなるほどアイテムの探索範囲を縮小できるため，効率良く時系列パターンを発見することが可能となる．

Yang らは，時系列パターンマイニングの処理において最も大きなコストはアイテムのカウントに関連する処理であるとし，アイテムの探索範囲の縮小を行うために時系列データベースに含まれるアイテムの最終出現位置を利用する LAPIN アルゴリズムの提案を行っている[6]．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上で，長さ k の時系列パターンより後の範囲をアイテムの探索範囲とし，その範囲内のアイテムの出現数のカウントを行う．この探索範囲は PrefixSpan の探索範囲と比べて，データベース上に同じアイテムが何度も出現するような場合に探索範囲を大きく縮小できる．さらに，Yang らは LAPIN アルゴリズムのアイテムの最終出現位置のリスト上で，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くことにより，出現回数の更新を行い，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[7]．LAPIN では長さ k の時系列パターン以降の全探索範囲でアイテムの出現数のカウントを行う必要があったのに対し，PAID では対応する長さ $k-1$ から k までの範囲に出現するアイテムを差し引き対象として処理を行えばよいため，より効率的に長さ $k+1$ の時系列パターンを発見できる．

一方，相関ルールマイニングではアルゴリズムをキャッシュ指向化して高速化を行う研究も行われている．Ghoting らは，頻出パターンマイニングアルゴリズム FP-growth[12] で利用されるデータ構造に対し，連続してデータにアクセス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により，空間的局所性の改良を行っている．また，CPU のキャッシュに収まるようにタイルといった単位でデータの分割を行い，フェッチしたタイルを無駄なく利用することにより時間的局所性の改良も行っている．このほかにも，マルチスレッドによりデータの再利用率を向上させ，最大で約 4.8 倍の速度向上が達成されている[10]．

PAID アルゴリズムは SPADE，PrefixSpan といった他の時系列パターンマイニングアルゴリズムよりも優れた処理効率であることが性能評価により示されている[7]．このため，本研究では改良対象として PAID アルゴリズムを選択し，CC-PAID の提案を行う．CC-PAID では CPU キャッシュの利用効率を改善するためにアクセスパターンの改良による時間的局所性の改善を行う．アクセスパターンに関しては，共通する接頭辞を基にメインメモリに収まるように探索範囲の分割を行う手法が SPADE により提案されている．CC-PAID においても，アクセスパターンの改良のために共通する接頭辞を利用するが，この共通する接頭辞は PAID のデータ構造の時間的局所性の改良を行うために利用され，CPU キャッシュを意識したものである．また，時間的局所性の改良に関しては，相関ルールマイニングの研究でタイルといったデータ構造を利用する提案が行われている．CC-PAID においては，PAID の特定のデータ構造に対して，共通する接頭辞を利用することにより一括したアクセスを可能とさせ，時間的局所性の改良を行う．

加えて，CC-PAID では処理時間の短縮を目的とした並列化が行われている．時系列パターンマイニングの並列アルゴリズムには分散共有メモリマシンを対象とした pSPADE アルゴリズムの提案が行われているが，本研究では近年普及し始めたマルチコアプロセッサを用いている．具体的には，PAID アルゴリズムで利用される 2 つのデータ構造を分割対象とし，それらを各プロセッサで処理するデータ並列を採用する．

3. 時系列パターンマイニング

時系列パターンマイニングは時系列データベースから最小支持度を満たす部分列を時系列パターンとして発見する手法である．時系列データベース SDB は複数の時系列データにより構成され，時系列データはアイテムセットにより構成される．

時系列データベースは，$SDB = \langle s_1, s_2, \ldots s_k \rangle$ と示され，s_t は時系列 ID（以降 SID）が付与された時系列データであり，SID 順に並んだ時系列データで構成される．時系列データは，$s = \langle is_1, is_2, \ldots, is_m \rangle$ で表され，is_n はアイテムセットであり，これを 1 つのトランザクションしてトランザクション ID（以降 TID）と呼ばれる時間情報または出現順序により識別され，TID 順に並んだアイテムセットで構成される．アイテムセットは，$is = (i_1, i_2, \ldots, i_c)$ であり，アイテムにより構成されている．時系列データに含まれる TID による順序関係を維持したアイテムの組合せを時系列データの長さ k の部分列 α としたとき，全時系列データの数に対して長さ k の部分列 α が出現する割合（長さ k の部分列 α の出現回数 / 全時系列データの数）または出現回数を支持度と呼ぶ．また，同じ SID 中に同一の部分列 α が複数存在しても，その SID 中の部分列 α の出現回数を 1 とする．

時系列パターンの発見では，マイニング開始時にユーザが決定した支持度を閾値となる最小支持度とし，時系列データベース内で長さ k の部分列 α の支持度を調べ，最小支持度以上の長さ k の部分列 α の支持度を満たすすべての長さ k の部分列 α を時系列パターンとする．また，時系列パターンの処理は itemset-extention step（以降 I-Step），sequence-extention step（以降 S-Step）に分けられる[3]．SDB 中に現れる is に関して，相関ルールマイニングを行い，is の中で最小支持度を満たす組合せを導出するのが I-Step であり，時系列パターンを導出するのが S-Step である．以下に時系列パターンマイニングを行った際の時系列パターンが出力される例を示す．例として，長さ 1 のアイテムセットにより構成されるトランザクションを含む時系列データベース（**図 1**）から最小支持度（回数）を 4 として，それを満たす時系列パターンの抽出例を**図 2** に示す．なお，図 1 の SID は時系列データの ID であり，例として SID 1 の時系列データは A，C，B，C，A，D の出現順で得られたトランザクション（それぞれ長さ 1 のアイテムセット）により構築されており，そのトランザクションが出現した時間をトランザクションの ID である TID で表している．次に図 2 の説明を行う．長さ 3 の時系列パターンに A → B → A : 4 があるが，これは長さ 3 の部分列 A → B → A（“→” はトランザクションの順序関係を示す）が図 1 の時系列データベース上でに 4 回出現することを意味する．

TID / SID	sequence data					
	1	2	3	4	5	6
1	A	C	B	C	A	D
2	A	D	B	A	C	
3	B	A	A	D	C	A
4	C	C	D	A	B	A
5	C	A	C	A	B	A

図 1　時系列データベース
Fig. 1　Sequence database.

length	Sequential pattern : Support
1	A:5, B:5, C:5, D: 4
2	A→A:5, A→B:4, A→C:4, B→A:5, C→A:4
3	A→B→A:4

図 2　最小支持度（回数）4 を満たす時系列パターン
Fig. 2　Sequential patterns where minimum support is 4.

また，支持度について追加説明を行うと，例として SID 3 の時系列データにアイテム A が 3 つ含まれているが，これらのアイテムは TID による順序関係があるため，その時系列データにおいてアイテム A の出現回数が 3 となるのではなく，部分列 A，A → A，A → A → A がそれぞれ 1 回出現することになる．長さ k の部分列には順序関係の制約があるため，1 つの時系列データに 1 回しか出現せず，支持度を出現回数とした場合，その最大値は時系列データ数となる．このため，例においては時系列データ数が 5 であるため，長さ k の部分列 α の出現回数も 5 回以上になることはない．

加えて，Prefixspan アルゴリズム[4] のような手法で長さ $k+1$ の時系列パターンを発見する場合の I-Step と S-Step の例を示す．時系列データ数が 1 つの時系列データベース (AB)CD があり（“()” は 1 つのトランザクションを示す），(AB) の TID は 1，C の TID は 2，D の TID は 3 となり，最小支持度（回数）を 1 と設定する．例として，長さ 1 の時系列パターン A が接頭辞となる長さ 2 の時系列パターンを発見する場合，時系列データベース上で長さ 1 の時系列パターン A 以降の範囲を投影データベース (_B)CD とする（(_B) は時系列パターン A と同じ TID に出現することを示す）．この投影データベース内で接頭辞が時系列パターン A である長さ 2 の部分列となるアイテムの支持度を調べる．I-Step では時系列パターン A と同じ TID である (_B) のアイテムの支持度を調べ，B は最小支持度を満たすため，長さ 2 の時系列パターン (AB) が発見される．S-Step では時系列パターン A の TID より大きい TID のトランザクションのアイテムとなる C と D の支持度を調べ，C と D ともに最小支持度を満たすため，長さ 2 の時系列パターン A → C，A → D が発見される．

また，本研究の提案は時系列パターンの発見処理となる S-Step に着目している．I-Step は相関ルールマイニングと同じであるため，次節からのアルゴリズムの説明等は S-Step に関するものとする．

```
PAID Algorithm
Input : A sequence database, minimum support
Output : All sequential patterns

Main
1. POS-DB, ILP-DB, FI_s, sup_list ← Scan_SDB()
2. For each item fi in FI_s
   2.1 Call Gen_(k+1)-patterns(fi, 0, sup_list)

Function Gen_(k+1)-patterns (k-sp, (k-1)-pbp_set, (k-1)-sup_list)
3. For each sequence sd
   3.1 k-pbp ← find_k_pbp((k-1)-pbp)
   3.2 S ← find_S_pos((k-1)-pbp)
   3.3 E ← find_E_pos((k)-pbp)
   3.4 while(S! = Null && S < E)
       3.4.1 k-sup_list[S.ItemID]--; //k-sup_list is a copy of (k-1)-sup_list.
       3.4.2 S++;
4. FI_s ← get_freq_item(k-sup_list)
5. For each item fi in FI_s
   5.1 (k+1)-sp ← extend_k-sp(f1, k-sp)
   5.2 Call Gen_(k+1)-patterns((k+1)-sp, k-pbp_set, k-sup_list)
```

図 3　PAID アルゴリズムの疑似コード
Fig. 3　PAID algorithm pseudo code.

3.1　時系列パターンマイニングアルゴリズム：PAID [7]

PAID アルゴリズムの概要および処理に関しての説明を疑似コードと例を用いて紹介する．また，PAID アルゴリズムの疑似コード（**図 3**）は参考文献 7) を参考にした．

PAID アルゴリズムでは，前処理として時系列データベースから position list of db（以降 POS-DB），item-last-position table（以降 ILP-DB），最小支持度を満たすアイテム集合（FI_s），ILP-DB 内のアイテムの支持度（sup_list）の調査および生成を行う（図 3-1）．POS-DB は SID の順で構成された POS-LIST の集合であり．POS-LIST は時系列データ上に出現するアイテムの TID をアイテムごとに昇順にまとめたものである．また，ILP-DB は SID の順で構成されたすべての ILP-list の集合であり，ILP-list はアイテムの最終出現位置（時系列データ上でアイテムが最後に出現する TID）と時系列データに含まれる最小支持度を満たすアイテムの組（TID，ItemID）により構成されており，最終出現位置（TID）をキーとして昇順にソートされている．ILP-list は長さ $k+1$ の部分列の支持度の調査に使用されるデータ構造である．

ここで，ILP-list と長さ $k+1$ の部分列の支持度の関係について説明する．まず，長さ

k の時系列パターンの出現位置（k 番目のアイテムの TID）が ILP-list 上のアイテムの最終出現位置（TID）より大きくなる位置以降の範囲を k-projILP-list としたとき，そのすべての k-projILP-list を k-projILPDB とし，k-projILPDB 内のアイテムの支持度を k-projILPDB_Sup とする．k-projILP-list 内のアイテムは接頭辞が長さ k の時系列パターンである長さ $k+1$ の部分列の $k+1$ 番目のアイテムとなる．このため，長さ $k+1$ の時系列パターンの発見処理において，長さ $k+1$ の部分列の支持度を調べるために k-projILPDB 内のアイテムの支持度を調べ，k-projILPDB_Sup を決定する．

k-projILPDB_Sup を決定する手順を示す．

(1) ある時系列データ上で長さ k 時系列パターンの TID（k 番目のアイテムの TID）を k-prefix border position（以降 k-pbp），その接頭辞となる長さ $k-1$ の時系列パターンの TID（$k-1$ 番目のアイテムの TID）を $(k-1)$-prefix border position（以降 $(k-1)$-pbp）とし，$(k-1)$-pbp を使い，POS-LIST の対応するデータから $(k-1)$-pbp より大きい TID となる k-pbp を調べる（図 3-3.1 に相当）．

(2) 対応する SID の ILP-list 上で $(k-1)$-pbp，k-pbp がアイテムの最終出現位置（TID）より大きくなる位置を調べ，$(k-1)$-pbp の位置を S，k-pbp の位置を E としたとき（図 3-3.2 と 3.3 に相当），ILP-list 上で S 以上で E までの範囲内のアイテムを調べる（図 3-3.4 に相当）．

(3) ILP-list 上で S の位置から始まる範囲を $(k-1)$-projILP-list としたとき，その $(k-1)$-projILP-list 上の E から始まる範囲が k-projILP-list となる．S から E までに出現するアイテムは $(k-1)$-projILP-list 上で長さ k の時系列パターンの位置（k 番目のアイテムの TID）より大きくなる位置以降で出現しなくなるアイテム（以降非出現となるアイテム）であるため，$(k-1)$-projILPDB_Sup で非出現となるアイテムの支持度から出現回数として 1 つ差し引く（図 3-3.4.1 に相当）．

すべての SID ごとで，(1)〜(3) の処理を行い，$(k-1)$-projILPDB_Sup の結果を k-projILPDB_Sup とする．k-projILPDB_Sup から最小支持度（minimum support）を満たすアイテムを調べ（図 3-4 に相当），それらのアイテムを処理対象としている長さ k の時系列パターンの後ろに連結して（図 3-5.1 に相当），長さ $k+1$ の時系列パターンが発見される（たとえば，長さ 1 の時系列パターンが A で最小支持度を満たすアイテムが A，B，C の場合，長さ 2 の時系列パターン A → A，A → B，A → C として発見される）．

また，図 3 の補足説明を行う．Gen_$(k+1)$-patterns の仮引数はそれぞれ，1 つの長さ k の時系列パターン（k-sp），第 2 引数は長さ $k-1$ の時系列パターンの pbp を SID ごとにまとめたセット（$(k-1)$-pbp_set），第 3 引数は $(k-1)$-projILPDB_Sup($(k-1)$-sup_list) を意味し，（図 3-3.4.1）の k-sup_list は引数で渡された $(k-1)$-projILPDB_Sup の複製である．

以下に図 1 の時系列データベースを対象に最小支持度（回数）を 4 と設定し，長さ 2 の時系列パターン A → B から長さ 3 となる時系列パターンの抽出例を示す．まず，**図 4** の POS-DB SID 1 の ItemID B の場所から時系列パターン A の pbp($(k-1)$-pbp) を用いて，時系列パターン A → B の pbp(k-pbp) を調べる．結果として時系列パターン A と A → B の pbp は (SID, TID) の組となる $(k-1)$-pbp $= \{(1,1)\}$，k-pbp $= \{(1,3)\}$ が取得される．$(k-1)$-pbp $= \{(1,1)\}$，k-pbp $= \{(1,3)\}$ を用いて，**図 5** の ILP-DB の SID 1 の ILP-list でそれらの TID より大きくなる位置を調べる（$(k-1)$-pbp，k-pbp より大きくなる位置を “↑” で示す）．また，図 5 の SID ごとのデータ内の要素は，(アイテムの最終出現位置 (TID), Item ID) の組である．以降では ILP-List 上で pbp よりも大きくなる位置を探す処理を位置合わせとする．位置を調べた結果から SID 1 の ILP-List 上で時系列パターン A → B より後に出現しないアイテムが B であることが分かる．ILP-DB 上で時系列パターン A-projILPDB_Sup からアイテム B を選択し，出現回数として 1 つ差し引く．このような処理をすべての SID を対象に行った結果を時系列パターン A → B-projILPDB_Sup として，それぞれのアイテムの支持度（**図 6**）を示す．この結果から，4 回以上出現するアイテムは A であることが分かり，時系列パターン A → B が接頭辞となる長さ 3 の時系列パターン A → B → A として発見される．

3.2　PAID のキャッシュ利用効率の問題点

時系列パターンマイニングでは発見された時系列パターンを辞書式順序木で表すことができる[3]．PAID アルゴリズムでは，長さ $k+1$ の時系列パターンの抽出処理の対象を決定する際，辞書式順序木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する．ここで，図 1 に対し，PAID により最小支持度（回数）を 4 としたときの時系列パターン A → B まで処理を行った場合のアクセスパターンを示す（**図 7**）．なお，枠で囲まれた時系列パターンは処理が行われたことを示す．この例では，深さ優先探索により，長さ k の時系列パターンとして A → B を 1 つ選択し処理を行うことにより，長さ 3 の時系列パターン A → B → A が発見される．

図 4 と図 5 の SID 3 を対象に，PAID のアクセスパターンによる ILP-list のキャッシュミスの原因と利用効率についての説明を行う．

初めに ILP-list のキャッシュミスに関する説明（**図 8**）を行う．図 8 は長さ 2 の時系列パ

SID	Item ID	Transaction ID
1	A	1, 5
	B	3
	C	2, 4
	D	6
2	A	1, 4
	B	3
	C	5
	D	2
3	A	2, 3, 6
	B	1
	C	5
	D	4
4	A	4, 6
	B	5
	C	1, 2
	D	3
5	A	2 , 4, 6
	B	5
	C	1, 3
	D	

図 4　データベース上の各アイテムの出現位置
Fig. 4　Position list of DB.

SID	Item Last Position List (Transaction ID, Item ID)
1	(3, B), (4, C), (5, A), (6, D)
2	(2, D), (3, B), (4, A), (5, C)
3	(1, B), (4, D), (5, C), (6, A)
4	(2, C), (3, D), (5, B), (6, A)
5	(3, C), (5, B), (6, A)

図 5　アイテムの最終出現位置を記録したテーブル
Fig. 5　Item last position table.

A

Item ID	A	B	C	D
Support	5	4	4	3

↓ -1　↓ -4　↓ -2　↓ -2

A→B

Item ID	A	B	C	D
Support	4	0	2	1

図 6　時系列パターン A と A → B の投影 ILPDB 内のアイテムの支持度
Fig. 6　Sequential pattern A and A → B's projected ILP-DB, and their item's support values.

ターン A → B の処理開始時の CPU キャッシュの状態を表し，破線は処理対象となっていることを示す．この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あり，これがキャッシュミスの原因の 1 つとなる．時系列パターンマイニングでは，基本的に SID に対して昇順にアクセスされるため，SID の番号が小さいと早くにキャッシュにフェッチされてしまう．このため，規模の大きいデータベース

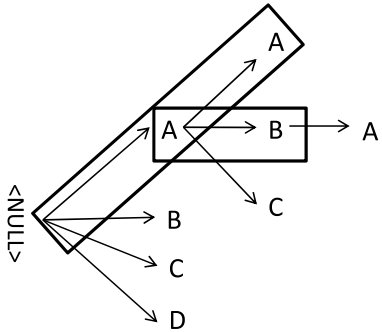


図 7　PAID のアクセスパターン
Fig. 7　Access pattern of PAID.

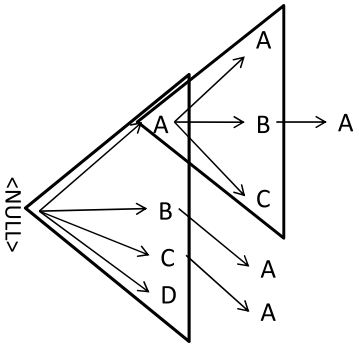


図 10　CC-PAID のアクセスパターン
Fig. 10　Access pattern of CC-PAID.

（例としては時系列データ数が非常に多い等）の処理では SID の番号が小さい ILP-list はキャッシュから外れてしまう可能性が高くなるため，キャッシュミスが増加しやすいと考えられる．

次に**図 9** を用いて ILP-list の利用効率に関する説明を行う．図 9 では，長さ 3 の時系列パターン A → B → A の処理と長さ 2 の時系列パターン A → C の処理を表している．時系列パターン A → B → A の処理では，図 4 から SID 3 の $(k-1)$-pbp（時系列パターン A → B の TID）は存在しないことが分かり，SID 3 の ILP-list にはアクセスが生じない．このため，時系列パターン A → B を処理したときの SID 3 の ILP-list が CPU キャッシュに存在していても時系列パターン A → B → A の処理時に再利用されない．それに対し，時系列パターン A → C では，図 4 の SID 3 から $(k-1)$-pbp（時系列パターン A の TID）は 2 であるために，ILP-list 上での非出現となるアイテムを調べるためのアクセスが生じる．ここで，時系列パターン A → B → A の処理終了時に SID 3 の ILP-list がキャッシュから外れていた場合を考えると，時系列パターン A → C の処理で再び SID 3 の ILP-list をフェッチする必要がでてくる．このため，PAID の従来のアクセスパターンでは CPU キャッシュに存在する ILP-List を効率的に再利用されない可能性がある．また，処理する時系列パターン長が長くなるほど，処理対象から外れる ILP-list が増える可能性があるため，最小支持度が小さい場合にこのような問題が起こりやすくなると考えられる．

ILP-list は k-projILPDB 内のアイテムの支持度を調べるために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，ILP-list に関連したキャッシュミスは全体の処理に対す

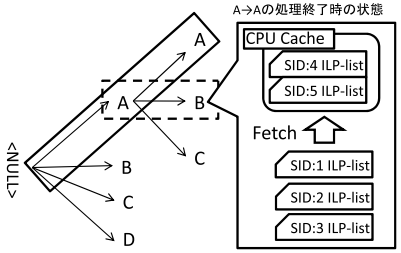


図 8　ILP-list のキャッシュミス
Fig. 8　Cache miss of ILP-list.

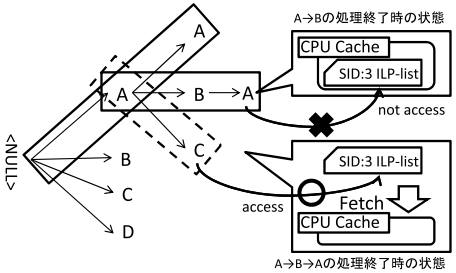


図 9　ILP-list のキャッシュ利用効率
Fig. 9　Cache utilization of ILP-list.

るボトルネックとなる可能性がある．これらの問題は CPU キャッシュにフェッチされたデータがすぐに再利用されないために起こりうると考えられ，従来のアクセスパターンでは時間的局所性が良くないといえる．この問題を解決するために，次章で CC-PAID を提案する．

4. CC-PAID

4.1 Common-Prefix-At-A-Time

ILP-list へのアクセスを考察すると，長さ $k-1$ の時系列パターンである共通する接頭辞（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 ILP-DB 上で同じとなる．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 2 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り，これらの時系列パターンの $(k-1)$-pbp は *common prefix*: A の TID となる．

本研究では各々の 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手法として Common-Prefix-At-A-Time の提案を行う．ここで，提案手法を用いた例として，図 7 と同様の条件で *common prefix* が A である長さ 2 の時系列パターンの処理が終了した際の時系列パターンを示す（**図 10**）．提案手法では，アクセスパターンとして *common prefix* を対象に深さ優先探索を適用させ，*common prefix* を含む複数の長さ k の時系列パターンを処理対象とする．*common prefix* を含む複数の長さ k の時系列パターンの処理範囲が同じになる性質を利用し，アクセスが生じた ILP-list に対し，*common prefix* を含む複数の長さ k の時系列パターンによる k-pbp の位置合わせと支持度からの差し引き処理が行われ，すべての SID の処理が終了した際に，*common prefix* を含む複数の長さ k の時系列パターンのそれぞれの長さ $k+1$ の時系列パターンが発見される．

```
CC-PAID Algorithm
Input : A sequence database, minimum support
Output : All sequential patterns

Main
do parallel {
1. POS-DB, ILP-DB, FI_s, sup_list ← Scan_SDB()
}
2. Call Gen_(k+1)-patterns(FI_s, 0, sup_list)

Function Gen_multi-(k+1)-patterns (multi-k-sp, (k-1)-pbp_set, (k-1)-sup_list)
3. For each sequence sd do parallel{
   3.1 For each k-sp in multi-k-sp
       3.1.1 multi-k-pbp ← find_multi-k-pbp((k-1)-pbp)
   3.2 S ← find_S_pos((k-1)-pbp)
   3.3 For each k-pbp in multi-k-pbp
       3.3.1 E ← find_E_pos((k)-pbp)
       3.3.2 while(S! = Null && S < E )
             3.3.2.1 (k-sp)'s k-sup_list[S.ItemID]--; //(k-sp)'s k-sup_list is a copy of (k-1)-sup_list.
             3.3.2.2 S++;
}
4. For each k-sp in multi-k-sp
   4.1 FI_s ← get_freq_item((k-sp)'s k-sup_list)
   4.2 multi-(k+1)-sp ← extend_k-sp(FI_s, k-sp)
   4.3 k-pbp_set ← get_k-pbp_set(multi-k-pbp_set)
   4.4 Call Gen_(k+1)-patterns(multi-(k+1)-sp, k-pbp_set, (k-sp)'s k-sup_list)
```

図 11　CC-PAID アルゴリズムの疑似コード
Fig. 11　CC-PAID algorithm pseudo code.

Common-Prefix-At-A-Time による CC-PAID アルゴリズムを疑似コード（**図 11**）と**図 12** を用いた例で説明する．図 12 は SID 1 の ILP-list を *common prefix*: A を選択して処理した例である．

(1)　（図 11）1. は前処理として PAID と同様に処理を行う．
(2)　（図 11）2. で関数 Gen_multi-$(k+1)$-patterns を呼び出す．CC-PAID では第 1 引数として複数の最小支持度を満たすアイテムを渡す．
(3)　（図 11）3. 各々の時系列データごとで 3.3.2.2 までの以下の処理を行う．
(4)　（図 11）3.1〜3.1.1 multi-k-sp は複数の長さ k の時系列パターンであり，各々の長さ k の時系列パターンごとに長さ k の時系列パターンの k 番目のアイテムとその k-pbp

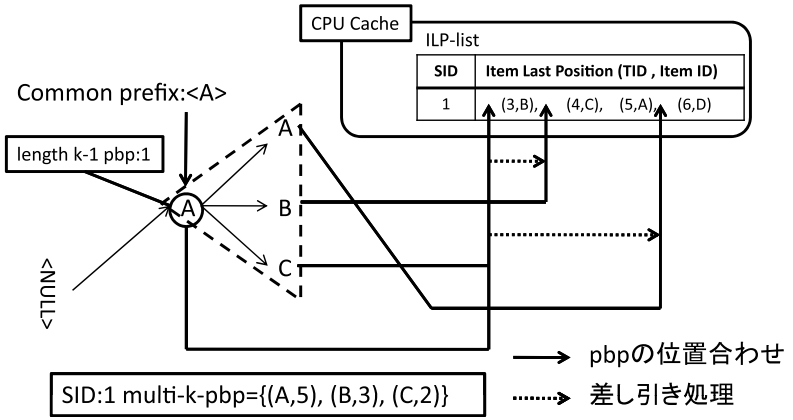


図 12　Common-Prefix-At-A-Time による ILP-list の処理
Fig. 12　Processing of ILP-list that applies Common-Prefix-At-A-Time.

の組を (ItemID, TID) として，multi-k-pbp に挿入していく．また，時系列データに対応するすべての multi-k-pbp をまとめたものを multi-k-pbp_set とする．

- 例では，図 4 の SID 1 の position list から $(k-1)$-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．結果として，SID 1 の multi-k-pbp= $\{(A,5),\ (B,3),\ (C,2)\}$ が得られる．

(5)　（図 11）3.2 PAID アルゴリズム同様の処理で S を取得.

- 例において，$(k-1)$-pbp（*common prefix*: A）は TID が 1 なので図 12 の ILP-list 上で $(3, \mathrm{B})$ の位置を $(k-1)$-projILPDB の開始位置とする．

(6)　（図 11）3.3〜3.3.2.2 multi-k-pbp の各々の k-pbp ごとに以下の処理を行う.
ILP-list 上で k-pbp より大きい TID の場所を探し，E を取得．ILP-list 上の S から E までの範囲のアイテムを非出現なアイテムとし，$(k-1)$-sup_list のコピーであり k-pbp に対応する長さ k の時系列パターンの k-sup_list((k-sp)'s k-sup_list) から出現回数として 1 つ差し引く.

- 例においては，まず時系列パターン A → A の k-pbp 5 より大きい位置を探し，その結果 $(6, \mathrm{D})$ の位置が得られ，非出現となるアイテムが A，B，C となることが分かる．時系列パターン A-projILPDB_Sup をコピーした時系列パターン A → A-projILPDB_Sup から A，B，C の出現回数として 1 つ差し引く．同様の処理を SID 1 の ILP-list に対し時系列パターン A → B，A → C で行う.

(7)　（図 11）4. 各々の長さ k の時系列パターンごとに 4.4 までの以下の処理を行う.

(8)　（図 11）4.1〜4.2 (k-sp)'s k-sup_list から最小支持度を満たすアイテム集合を取得．それらのアイテムを k-sp の後ろに連結し，$(k+1)$-sp とした各々のセットを multi-$(k+1)$-sp とする.

- 例においては，時系列パターン A → B で最小支持度を満たすアイテム A が発見され，長さ $k+1$ として時系列パターン A → B → A が発見される.

(9)　（図 11）4.3 multi-k-pbp_set から時系列データごとの k-pbp を取得し，k-pbp_set としてまとめる.

- 例においては，時系列パターン A → B では図 12 の SID 1 の multi-k-pbp からアイテム B を選択して 3 を取得，同様に以降の SID からも k-pbp を取得してまとめる.

(10)　（図 11）4.4 関数 Gen_multi-$(k+1)$-patterns を呼び出す.

- 例においては，時系列パターン A → A では長さ $k+1$ の時系列パターンが発見されなかったので，深さ優先探索に従い，時系列パターン A → B を *common prefix*: A → B として選択し，次の長さ $k+1$ の発見処理に移行する.

ここで実装についての補足説明を行う．$(k-1)$-pbp が NULL の場合に，対応する時系列データに関する処理として PAID では図 3-3.1〜3.4.2，CC-PAID では図 11-3.1〜3.3.2.2 までの処理を回避できる．また，PAID では図 3-3.3 で得られる ILP-list 上の位置 E を SID との組でまとめた candidate border index set としており，次の長さ $k+1$ の時系列パターン発見処理時に，図 3-3.2 の S として利用できる．さらに，図 3-3.3 の処理で ILP-list 上の位置 E を探す範囲として，開始地点を S とした $(k-1)$-projILP-list の範囲で位置 E を探すことができる．CC-PAID でも，ILP-list の位置 E を multi-k-pbp_set と同様に multi-k-cbi_set として保持することにより，（図 11）4.3 の処理と同様に candidate border index set を取得し，PAID と同じように candidate border index set を利用することが可能である.

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.

4.2 CC-PAID の並列化

時系列パターンマイニングでは支持度の更新に関する処理を時系列データごとに独立して処理することが可能であり，CPU コア数に応じて時系列データベースを分割することによる並列化が可能である．また，大規模なデータベースでは時系列データ数が膨大になる可能性がありデータ並列性が高いと考えられる．そこで，本研究では並列化手法として分割された時系列データベースによるデータ並列を採用し，並列処理による処理時間の短縮も試みる．

CC-PAID では do parallel で示す図 11-1. の前処理と図 11-3.〜3.3.2.2 の部分が並列処理可能である．PAID では図 3-1 と図 3-3〜3.4.2 の部分がこれに相当する．

PAID および CC-PAID における並列化では，時系列データベースの時系列データ数を長さ $k+1$ の時系列パターン抽出処理を行うスレッド数で分割し，それぞれのスレッドで処理対象とする SID の範囲を決定する．たとえば，時系列データ数が 100，実行スレッド数が 2 の場合，スレッド A は SID 1〜50，スレッド B は SID 51〜100 までが割り当てられ，POS-DB および ILP-DB の対応する SID の範囲に対してそれぞれのスレッドでアクセスされる．

また，支持度のデータ（CC-PAID では (k-sp)'s k-sup_list）からの差し引き処理では排他制御が必要となる．このため，時系列データごとの差し引き処理では多くの排他制御が必要となるため，実装においては差し引き処理を行う処理の部分（CC-PAID では図 11-3.3.2.1）で，非出現となるアイテムの出現数のカウントを行い，スレッドの処理対象とする SID の処理のすべてが終了した際，支持度のデータから排他制御により差し引き処理を行う．これにより排他制御の回数を抑えることが可能となる．

5. 評 価 実 験

本章では，既存の時系列パターンマイニングアルゴリズム PAID と提案手法 Common-Prefix-At-A-Time を応用した CC-PAID を評価対象とし，特徴の違うデータセットを用いて，処理時間とキャッシュミスの評価を行う．また，本研究では処理時間の短縮を目的として，並列処理時の処理時間およびキャッシュミスの評価も行う．PAID の並列化に関しては図 3-1. と図 3-3.〜3.4.2 の部分を並列化したものを利用した．なお，本研究は S-Step に着目しているため，PAID および CC-PAID は S-Step 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．また，キャッシュミスの測定には Intel VTune Amplifier XE 2011 を用い，実験環境において最もキャッシュミスによるレイテンシの大きい L3 のキャッシュミスの見積りの調査を行った．実験で用いるデータセットは IBM Data Generator により生成を行った．ここで，特徴の違うデータセットを用いた実験を行うため，基準とするデータセットの生成時に使用したパラメータを**表 2** に示す．なお，最小支持度には出現割合を使用する．たとえば，最小支持度 0.1 としたとき，全時系列データ数の 10%を表す．

表 1　実験環境
Table 1　Experimental enviroment.

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

表 2　データセット D_1 のパラメータ
Table 2　Parameters of data set D_1.

時系列データ数	50,000
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

5.1 処理時間とキャッシュミスの評価

基準とするデータセット D_1（表 2）に対し，時系列データ数を 2 倍の 100,000 にしたデータセット D_2，時系列データ内の平均トランザクション数を 2 倍の 100 にしたデータセット D_3，アイテムの種類数を 2 倍の 800 にしたデータセット D_4 を用い，それぞれのデータセットで最小支持度を変更した際の処理時間と L3 キャッシュミスの見積りを調べた．加えて，各データセットで使用した最も低い最小支持度を利用した際のメモリ使用量も調べた．以下に，それぞれのデータセットごとの計測結果を示す．

- （**図 13**）データセット D_1：最小支持度 0.3，0.35，0.4，0.45
  - 処理時間
    処理時間に関しては PAID に対して，CC-PAID では最小支持度 0.3 のとき，約 1.99 倍の高速化，最小支持度 0.35 のとき，約 1.94 倍の高速化，最小支持度 0.4 のとき，約 1.9 倍の高速化，最小支持度 0.45 のとき，約 1.77 倍の高速化となった．
  - キャッシュミス
    キャッシュミスに関しては PAID に対して，CC-PAID では最小支持度 0.3 のとき，約 61%の削減，最小支持度 0.35 のとき，約 60%の削減，最小支持度 0.4 のとき，約 58%の削減，最小支持度 0.45 のとき，約 56%の削減となった．

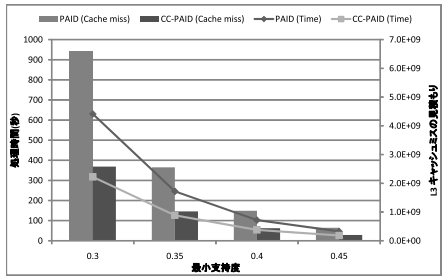


図 13　D_1：処理時間と L3 キャッシュミスの見積り
Fig. 13　D_1: Processing time and estimate of L3 cache miss.

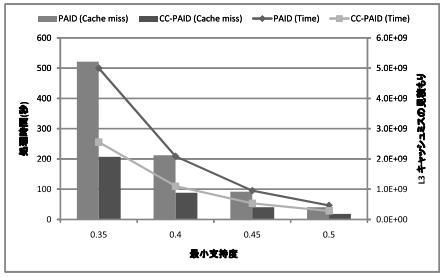


図 14　D_2：処理時間と L3 キャッシュミスの見積り
Fig. 14　D_2: Processing time and estimate of L3 cache miss.

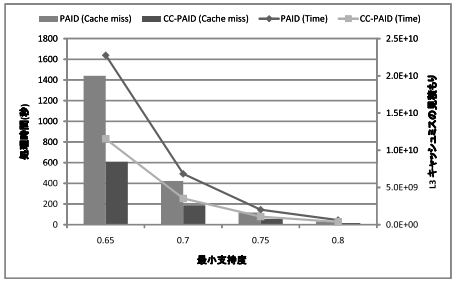


図 15　D_3：処理時間と L3 キャッシュミスの見積り
Fig. 15　D_3: Processing time and estimate of L3 cache miss.

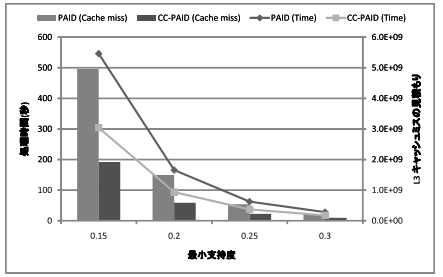


図 16　D_4：処理時間と L3 キャッシュミスの見積り
Fig. 16　D_4: Processing time and estimate of L3 cache miss.

- （**図 14**）データセット D_2：最小支持度 0.35，0.4，0.45，0.5
  - 処理時間
    処理時間に関しては PAID に対して，CC-PAID では最小支持度 0.35 のとき，約 1.95 倍の高速化，最小支持度 0.4 のとき，約 1.9 倍の高速化，最小支持度 0.45 のとき，約 1.79 倍の高速化，最小支持度 0.5 のとき，約 1.66 倍の高速化となった．
  - キャッシュミス
    キャッシュミスに関しては PAID に対して，CC-PAID では最小支持度 0.35 のとき，約 60%の削減，最小支持度 0.4 のとき，約 58%の削減，最小支持度 0.45 のとき，約 56%の削減，最小支持度 0.5 のとき，約 55%の削減となった．
- （**図 15**）データセット D_3：最小支持度 0.65，0.7，0.75，0.8
  - 処理時間
    処理時間に関しては PAID に対して，CC-PAID では最小支持度 0.65 のとき，約 1.97 倍の高速化，最小支持度 0.7 のとき，約 1.94 倍の高速化，最小支持度 0.75 のとき，約 1.85 倍の高速化，最小支持度 0.8 のとき，約 1.67 倍の高速化となった．
  - キャッシュミス
    キャッシュミスに関しては PAID に対して，CC-PAID では最小支持度 0.65 のとき，約 58%の削減，最小支持度 0.7 のとき，約 56%の削減，最小支持度 0.75 のとき，約 52%の削減，最小支持度 0.8 のとき，約 49%の削減となった．
- （**図 16**）データセット D_4：最小支持度 0.15，0.2，0.25，0.3
  - 処理時間
    処理時間に関しては PAID に対して，CC-PAID では最小支持度 0.15 のとき，約 1.8 倍の高速化，最小支持度 0.2 のとき，約 1.77 倍の高速化，最小支持度 0.25 のとき，約 1.69 倍の高速化，最小支持度 0.3 のとき，約 1.61 倍の高速化となった．
  - キャッシュミス
    キャッシュミスに関しては PAID に対して，CC-PAID では最小支持度 0.15 のとき，約 61%の削減，最小支持度 0.2 のとき，約 61%の削減，最小支持度 0.25 のとき，約 58%の削減，最小支持度 0.3 のとき，約 56%の削減となった．

処理時間とキャッシュミスの考察としては，計測したすべてのデータセットと最小支持度で，PAID に対し CC-PAID の高速化（約 1.61 倍〜1.99 倍）とキャッシュミスの削減（約 49%〜61%）が確認され，高速化とともにキャッシュミスも削減される傾向が確認できるため，キャッシュミスの軽減が処理時間の短縮に大きく影響を与えたと考えられる．また，最小支持度を下げていくと PAID に対して，より高速化されることが確認できる．これは最小支持度を下げることにより発見される時系列パターン数が増加し，それにともない，PAID でのキャッシュミスが増加することにより，CC-PAID のキャッシュ利用効率の効果が顕著になるためと考えられる．

次に，メモリ使用量の調査結果を示す．

- データセット D_1：最小支持度 0.3
  PAID：約 330 MB，CC-PAID：約 530 MB

- データセット D_2：最小支持度 0.35
  PAID：約 660 MB，CC-PAID：約 980 MB
- データセット D_3：最小支持度 0.65
  PAID：約 420 MB，CC-PAID：約 600 MB
- データセット D_4：最小支持度 0.15
  PAID：約 520 MB，CC-PAID：約 890 MB

PAID に対し CC-PAID では 3 割〜4 割程のメモリ使用量が増加した．これは Common-Prefix-At-A-Time により，共通する接頭辞を持つ時系列パターンごとの処理に必要な支持度のデータ等を保持しなくてはならないためと考えられる．しかしながら，本研究で利用したデータセットと最小支持度に関した処理速度の向上に対し，メモリ使用量の増加する割合を考えると，処理速度が向上する割合の方が大きいと考えられるため，CC-PAID は有用であると思われる．また，Common-Prefix-At-A-Time では CPU キャッシュにフェッチされたデータを最も効率良く利用するために共通する接頭辞を持つ時系列パターンすべてを処理対象としているが，処理対象とする時系列パターン数を調整することにより，メモリ使用量を抑えることも可能と考えられる．しかし，処理対象とする時系列パターン数を調整するとキャッシュミスが増加する可能性が高くなることも考えられるため，処理速度向上の観点からは，共通する接頭辞を持つ時系列パターンすべてを処理対象とするのが望ましいと考えられる．

5.2　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 0.7 と設定し，スレッド数を 1，2，4，6 と増やしたときの処理時間の計測結果を**図 17** に示す．また，利用するデータセットは，基準とするデータセット D_1 に対し時系列データ数を 2 倍の 100,000，時系列データ内の平均トランザクション数を 2 倍の 100 にしたデータセット D_5 とする．処理時間に関しては PAID に対し，スレッド数 2 のとき，1.92 倍の高速化，スレッド数 4 のとき，1.92 倍の高速化，スレッド数 6 のとき，1.97 倍の高速化が確認された．スケーラビリティとしては，スレッド数 1 に対してスレッド数 6 のとき，PAID と CC-PAID ではともに 4.3 倍の高速化が行われた．

次に，最小支持度を 0.7 と設定し，スレッド数を 2，4，6 と増やしたときの各 CPU コアの L3 キャッシュミスの見積りとそれらを合計した値を**図 18** に示す．なお，各 CPU コアの L3 キャッシュミスの見積りを少ない順に整列させ，図 18 の core id の順に対応させている．

結果として合計値から，PAID に対して CC-PAID では L3 キャッシュミスに関してスレッド数 2 のとき，56%の削減，スレッド数 4 のとき，55%の削減，スレッド数 6 のとき，56%の

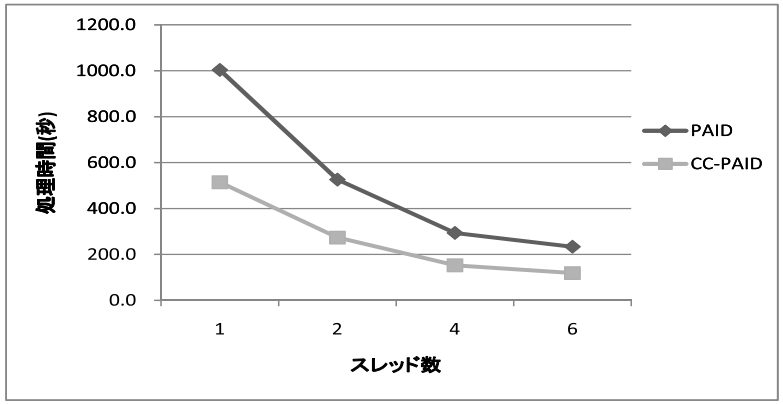
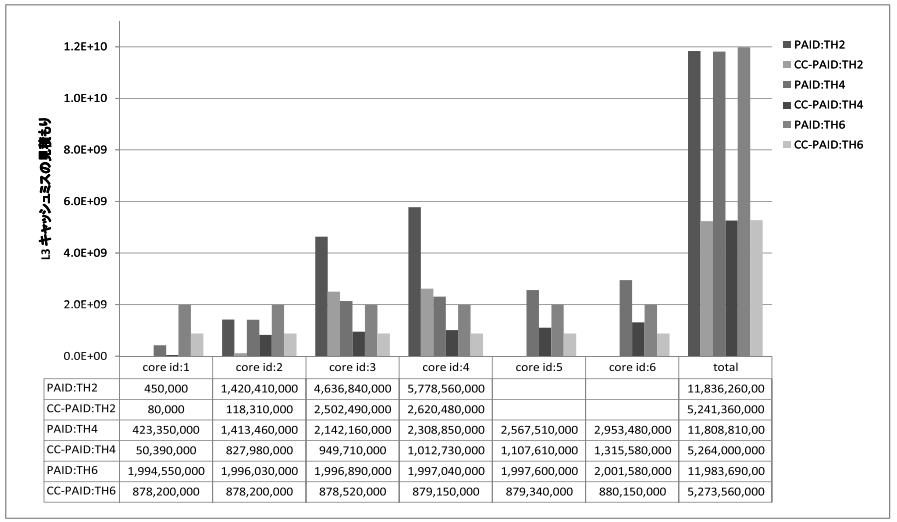


図 17　スレッド数増加時の処理時間
Fig. 17　Processing time when the number of threads increases.



	core id:1	core id:2	core id:3	core id:4	core id:5	core id:6	total
PAID:TH2	450,000	1,420,410,000	4,636,840,000	5,778,560,000			11,836,260,00
CC-PAID:TH2	80,000	118,310,000	2,502,490,000	2,620,480,000			5,241,360,000
PAID:TH4	423,350,000	1,413,460,000	2,142,160,000	2,308,850,000	2,567,510,000	2,953,480,000	11,808,810,00
CC-PAID:TH4	50,390,000	827,980,000	949,710,000	1,012,730,000	1,107,610,000	1,315,580,000	5,264,000,000
PAID:TH6	1,994,550,000	1,996,030,000	1,996,890,000	1,997,040,000	1,997,600,000	2,001,580,000	11,983,690,00
CC-PAID:TH6	878,200,000	878,200,000	878,520,000	879,150,000	879,340,000	880,150,000	5,273,560,000

図 18　スレッド数増加時の L3 キャッシュミスカウントの見積り
Fig. 18　L3 cache miss counts when the number of threads increases.

削減が確認された．図 18 から CC-PAID では，各 CPU コアで生じる L3 キャッシュミスが PAID に対して削減されていることが分かる．また，シングルスレッド時と同様の高速化が行われているため，CC-PAID は並列処理時も有効的にキャッシュミスの削減による高速化を実現できると考えられる．

6. まとめと今後の課題

本論文では，PAID アルゴリズムで利用される ILP-list に関する CPU キャッシュの利用効率に着目し，処理対象を，共通する接頭辞を含む複数の時系列パターンとすることにより，CPU キャッシュにフェッチされた ILP-list の再利用率を向上させる，時間的局所性の改良手法として Common-Prefix-At-A-Time の提案を行った．また，データ並列を用いた並列化による処理時間の短縮も行い，処理時間と CPU キャッシュミスの計測から CC-PAID の有効性を確認した．処理時間では PAID に対し，CC-PAID は最大で 2 倍近い高速化が確認された．キャッシュミスに関しても，PAID に対して CC-PAID では最大で 6 割程度のキャッシュミスの削減が確認された．さらに，CC-PAID では並列化により，スレッド数を 1 から 6 に増やした際に約 4.3 倍の高速化が確認された．

今後の課題としては，時系列パターンマイニングでは処理するデータベースの時系列データ数が多い場合があり，さらに反復的なアクセスが行われるため，Cell Broadband Engine や GPGPU といったメニーコアプロセッサによる処理が効果的であると考えられ，他のアーキテクチャによる処理時間の短縮方法の提案があげられる．また，性能評価に関しては人工データ以外にも実データでの評価を行う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．

参 考 文 献

1) Srikant, R. and Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements, *Proc. 5th International Conference on Extending Database Technology: Advances in Database Technology*, *EDBT '96*, pp.3–17, London, UK, Springer-Verlag (1996).
2) Zaki, M.J.: Efficient enumeration of frequent sequences, *Proc. 7th International Conference on Information and Knowledge Management*, *CIKM '98*, pp.68–75, New York, NY, USA, ACM (1998).
3) Ayres, J., Gehrke, J., Yiu, T. and Flannick, J.: Sequential PAttern mining using a bitmap representation, *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *KDD '02*, pp.429–435, New York, NY, USA, ACM (2002).
4) Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.-C.: Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach, *IEEE Trans. Knowledge and Data Engineering*, Vol.16, pp.1424–1440 (2004).
5) Wang, J., Asanuma, Y., Kodama, E., Takata, T. and Li, J.: Mining Sequential Patterns More Efficiently by Reducing the Cost of Scanning Sequence Databases, 情報処理学会論文誌，Vol.47, No.12, pp.3365–3379 (2006-12-15).
6) Yang, Z., Wang, Y. and Kitsuregawa, M.: LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases, *DASFAA*, pp.1020–1023 (2007).
7) Yang, Z., Kitsuregawa, M. and Wang, Y.: PAID: Mining Sequential Patterns by Passed Item Deduction in Large Databases, *Proc. 10th International Database Engineering and Applications Symposium*, pp.113–120, Washington, DC, USA, IEEE Computer Society (2006).
8) Zaki, M.J.: Parallel sequence mining on shared-memory machines, *Journal of Parallel and Distributed Computing*, pp.401–426, Springer-Verlag (2001).
9) 高木　允，田村慶一，周藤俊秀，北上　始：Modified PrefixSpan 法の並列化と動的負荷分散手法，情報処理学会論文誌：数理モデル化と応用，Vol.46, No.10, pp.138–152 (2005).
10) 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, *Proc. 31st International Conference on Very Large Data Bases*, *VLDB '05*, pp.577–588, VLDB Endowment (2005).
11) Wulf, W.A. and McKee, S.A.: Hitting the memory wall: Implications of the obvious, *SIGARCH Comput. Archit. News*, Vol.23, pp.20–24 (1995).
12) Han, J., Pei, J. and Yin, Y.: Mining frequent patterns without candidate generation, *Proc. 2000 ACM SIGMOD international conference on Management of data*, *SIGMOD '00*, pp.1–12, New York, NY, USA, ACM (2000).

(平成 22 年 12 月 20 日受付)
(平成 23 年　4 月 12 日採録)

(担当編集委員　原田 リリアン)

松原　裕貴

2007 年岩手県立大学ソフトウェア情報学部ソフトウェア情報学科卒業．2009 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年同大学院同研究科博士後期課程入学，現在に至る．

宮崎　　純（正会員）

奈良先端科学技術大学院大学情報科学研究科准教授．博士（情報科学）．1992 年東京工業大学工学部情報工学科卒業．1997 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同大学助手を経て，2003 年より現職．2000〜2001 年テキサス大学アーリントン校客員研究員．2003〜2007 年科学技術振興機構さきがけ研究員．高性能・高機能データベースならびに情報検索の研究に従事．電子情報通信学会，日本データベース学会，ACM，IEEE CS 各会員．

藤澤　　誠（正会員）

2003 年静岡大学工学部機械工学科卒業．2005 年同大学大学院理工学研究科修士課程修了．2008 年同博士課程修了．同年奈良先端科学技術大学院大学情報科学研究科助教．2011 年筑波大学大学院図書館情報メディア研究科助教．博士（工学）．CG，物理シミュレーション等の研究に従事．ACM，IEEE 各会員．

天野　敏之

2000 年大阪大学大学院博士後期課程修了．同年名古屋工業大学助手．2007 年奈良先端科学技術大学院大学助教．2011 年山形大学工学部准教授．その間，2009 年バウハウス大学客員研究員．博士（工学）．パターン認識，コンピュータビジョン，拡張現実感の研究に従事．電子情報通信学会学術奨励賞（2000 年），画像の認識・理解シンポジウムインタラクティブセッション優秀賞（2006 年，2007 年），ベストデモセッション賞（2008 年，2010 年）受賞．電子情報通信学会シニア会員，PRMU 研究会専門委員会委員，IEEE，バーチャルリアリティ学会，ヒューマンインタフェース学会各会員．

加藤　博一（正会員）

1986 年大阪大学基礎工学部制御工学科卒業．1988 年同大学大学院修士課程修了．1989 年同大学基礎工学部助手．1996 年講師．1998 年ワシントン大学客員研究員．1999 年広島市立大学情報科学部助教授．2003 年大阪大学大学院基礎工学研究科助教授．2007 年より奈良先端科学技術大学院大学情報科学研究科教授．博士（工学）．拡張現実感，ヒューマンインタフェース，メディア情報処理，エンタテインメントコンピューティングの研究に従事．ヒューマンインタフェース学会，日本 VR 学会，電子情報通信学会，ACM，IEEE 等各会員．