Vol.2010-DBS-151 No.9
2010/11/12

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

松原裕貴[†1]　宮崎純[†1]　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多くの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, 時系列パターンマイニングではデータベース全体に対して非常に多くの反復的なアクセスが生じる傾向にあり, キャッシュミスが発生しやすいと考えられる. そのため, 大規模なデータベースや小さな閾値を利用した場合のキャッシュミスによるレイテンシは, 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ある. このため, 処理対象のアクセスパターンとデータ構造等の改良による特定のデータ構造に対するキャッシュミスの軽減手法を提案する.

Cache-Conscious Parallel PAID: A Parallel Sequential Pattern Mining Algorithm with the Effective Use of Cache Memory

Yuki Matsubara ,[†1] Jun Miyazaki [†1]
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 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 cache misses can reduce.

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保持した状態の相関ルール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挙げられる. ある百貨店の一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相関ルールとして, 例えば「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」といったことがわかったとする. この結果を元に, 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. この他にも 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サポート値とデータベースの規模が挙げられる. 最小サポート値とは, 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し, 処理時間が増加していく. また, パターンの抽出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, 小さな最小サポート値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)2)3)4)5)6)], 並列化による処理時間の改善[7)8)] 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, 実際の利用において処理時間に制約がある場合は利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サポート値を大きく設定することは効果的だが, この場合は有用

†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

とされる時系列パターンを取り逃がしてしまうといった問題も生じる．その他にも抽出されたパターンを利用して更に他の処理を行う場合等から, 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, キャッシュミスやデータ構造の再構築によるメインメモリへのアクセス回数の軽減といった CPU キャッシュの利用効率による処理時間の短縮は重要な課題であると考えている．

本研究では, 既存の PAID アルゴリズムの CPU キャッシュ利用効率に着目した改良と並列化による処理速度の向上手法の提案を行う．CPU キャッシュに関した提案は”Common-prefix-at-a-time”と”Rebuild-ILP”の２つである．まず１つ目の改良は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まとめてアクセスを行うことにより, CPU キャッシュにフェッチされたデータの再利用率を向上させる．２つ目は時系列パターンの処理を進めていくに従って探索の対象から外れるデータが有り，１つめの提案を応用することにより探索の対象となるデータのみで探索範囲を高速に再構築する．探索範囲の縮小によりデータへのアクセス回数を減らすことが期待できる．

2. 関 連 研 究

Srikant らは, 長さ k の時系列パターン同士で長さ $k+1$ の時系列パターンとなる可能性のある候補を生成し, データベース上での出現数を調べることにより長さ $k+1$ の時系列パターンの決定を行う GSP アルゴリズムの提案を行っている[1]．Zaki らは時系列データの部分列が出現する時系列データの ID と出現した時間のセットを垂直に表現した id-list を用い, *lattice* により, 長さ k の時系列パターンの id-list の *join* を行い, *join* した結果から閾値を満たすものを長さ $k+1$ の時系列パターンとする SPADE アルゴリズムの提案を行っている[2]．大規模なデータベースの探索処理では入出力を小さくすることが重要と考え, メインメモリ内で独立して処理を行えるように, 探索範囲となる *lattice* を分割する方法等も提案している．さらに, Zaki らは分散共有メモリマシンを対象に並列化等の改良を行った pSPADE アルゴリズムの提案も行っている[7]．

Pei らは時系列パターンを伸ばすことにより探索範囲を縮小させる PrefixSpan アルゴリズムの提案を行っている[3]．これは長さ k の時系列パターンがデータベース上のどこに現れるか調べ, その時系列パターンより後に出現するデータを投影データベースと呼び投影データベース内でアイテムの探索処理を行い, 閾値を満たすアイテムを長さ 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．時系列パターンが長くなるほど探索範囲を縮小できるため, 効率よく時系列パターンを発見できる．

Yang らは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ことにより, 探索範囲の縮小を行う LAPIN アルゴリズムの提案を行っている[5]．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．更に 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を用いて, 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, 出現しないアイテムに対応する長さ k の時系列パターンの出現数から差し引いて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[6]．処理途中の時系列パターンの出現回数を利用することにより, 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 までの範囲で出現するアイテムの差し引き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．

Ghoting らは, 頻出パターンマイニングアルゴリズム FP-growth[9] 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, 空間的参照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データの分割を行い, フェッチしたデータを無駄なく利用することにより時間的参照局所性の改良も行っている．この他にも, 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]．

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時系列パターンを発見する手法で

ある. 時系列データベースとは, 複数の時系列データ $SDB = \{s_1, s_2, \ldots s_k\}$ により構成されており, 時系列データはトランザクションの列 $seq = \{t_1, t_2, \ldots, t_n\}$ により構成される. さらに, トランザクションはアイテムセット $I = \{i_1, i_2, \ldots i_c\}$ により構築されている. また, サポートとは時系列データに含まれる部分列の時系列データベース全体に対する出現頻度または回数を指し, 最小サポート値 ($minsup$) を閾値として, 出現順序を保持した状態の部分列 ($subsequence$) でこれを満たすものを相関ルールと見なし, 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 (I-Step), sequence-extention step (S-Step) に分けられる[12]. I-Step ではアイテムセット内でパターンを伸ばし, S-Step では時系列データ内のトランザクションのパターンを伸ばす処理を行う.

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) から最小サポート値 4 を満たす時系列パターンの抽出例 (図 2) を示す. 図 1 の時系列データについて説明する. 時系列データの ID は sequence id(以降 SID と略す) として表わされており, 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アイテムを含むトランザクションにより構築されており, トランザクションが出現した時間を transaction id (以降 TID と略す) で表わしている. 次に図 2 をの説明を行う. 長さ 2 の時系列パターンに A → A:5 があるが, これは長さ 2 の時系列パターン A → A が図 1 の時系列データベースに 5 回出現することを示している.

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

図 1 時系列データベース

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

図 2 時系列パターン

本節では, PAID アルゴリズムの説明も行う. PAID ではアイテムの探索に LAPIN アルゴリズムで使われている item-last-position list(以降 ILP-list と略す) を利用している. ILP-list は時系列データに含まれるアイテムとそのアイテムの最終出現位置を記録したセットにより構成されており, 最終出現位置でソートされている. LAPIN アルゴリズムでは, ILP-list のデータベース (以降 ILP-DB と略す) 上の長さ k の時系列パターンより後の範囲 (以降 k-projdb と略す) をアイテムのカウント対象として利用する. PAID では長さ k の時系列パターンより後の ILP-list の範囲 (以降 k-projlist と略す) に出現しないアイテムを調べる. k-projlist に出現しないということは, 長さ $k+1$ の時系列パターンとなる可能性のある長さ $k+1$ の部分列とならないため, 長さ $k-1$ の時系列パターンより後の ILP-DB の範囲 (以降 $k-1$-projdb と略す) に含まれる各アイテムのサポートから差し引くことが可能である. 全ての SID に対応する ILP-list を対象に処理を行うことで, 結果として $k+1$ の候補パターンのサポートが得られ, 最小サポート値を満たすものを $k+1$ の時系列パターンとして発見する.

長さ k の時系列パターンから長さ $k+1$ の時系列パターンを発見する手順の説明を行う.

(1) 時系列データ上の長さ $k-1$ と k の時系列パターンの出現時間 (TID) を調べる (長さ $k-1$ の時系列パターンは長さ k の時系列パターンに含まれる接頭辞 (以降 $prefix$ と略す) である). この出現時間を $k-1$, k-prefix border position(以降 $k-1$-pbp, k-pbp と略す) とする.

(2) ILP-list 上で $k-1$,k-pbp の位置合わせを行い, $k-1$-pbp より後, k-pbp より前の ILP-list 上の範囲にあるアイテムを k-projlist に出現しないアイテムとする.

(3) $k-1$-projdb 内のアイテムのサポートから非出現となったアイテムを差し引く.

(4) 1 から 4 までの処理を全 SID に対して行い, $k-1$-projdb 内のアイテムで最小サポート値を満たすものを長さ k の時系列パターンに連結し, $k+1$ の時系列パターンとして発見する.

例として図 1 の時系列データベースを対象に最小サポート値 (回数) 4 と設定し, 長さ 2 の時系列パターン A → A から長さ 3 となる時系列パターンの抽出例を示す. まず, position list of db(以降 pos-db と略す) と呼ばれる各時系列データのトランザクションの出現時間 (TID) を記録したデータベース (図 3) から時系列パターン A の pbp($k-1$-pbp) と時系列パターン A → A の pbp(k-pbp) を全ての SID を対象として調べる. 結果として時系列パターン A の pbp は SID:TID の組となる $k-1$-pbp set= $\{1:1,\ 2:1,\ 3:2,\ 4:3,\ 5:2\}$ として表わされる. 同様に時系列パターン A → A も k-pbp set= $\{1:5,\ 2:4,\ 3:3,\ 4:5,\ 5:4\}$ と表わされる. 次に $k-1$ と k-pbp set を用いて, item-last-position table(以降 ILP-DB と略す) (図 4) の ILP-list で位置合わせを行う. 図 4 では $k-1$,k-pbp の位置を↑で示す. pos-db(図 3) の SID 2 では時系列パターン A の pbp は 1, 時系列パターン A → A の pbp は 4 であり, 図 4 の SID 2 から ILP-list での位置合わせの結果として, SID 2 では時系列パ

ターン A → A の以降の処理において Item A, B が出現しないことがわかり, 差し引き対象とされる. また, 全 SID を対象に行った結果として, ILP-DB 上で時系列パターン A と A → A より後の範囲のアイテムのサポート (図 5) を示す. この結果から, 4 回以上出現する Item は D であるため, 時系列パターン A → A が prefix となる長さ 3 の時系列パターンは A → A → D となる. なお, 長さ $k+1$ の複数の時系列パターンの抽出処理において, 深さ優先探索により, 単数の長さ k の時系列パターンを対象に時系列パターン長を伸ばしていく.

4. 提案方式

4.1 Common-prefix-at-a-time の提案

PAID アルゴリズムにおいて, ILP-list はサポートからの差し引き対象を調べるために再帰的なアクセスが生じるデータ構造である. このため, ILP-list のキャッシュミスは全体の処理に対するボトルネックとなる可能性がある. ILP-list へのアクセスを考察すると, 共通する *prefix* (以降 *common prefix* と略す) を含む長さ k の時系列パターンが複数存在する場合, それらの処理対象となる ILP-list の SID と探索範囲は同じである. 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 に対応する位置以降が非出現となるアイテムを探索するための範囲となるからである. PAID では深さ優先探索で長さ $k+1$ の時系列パターンの発見を行うための処理対象として, 一つの長さ k の時系列パターンを選択するため, 他の *common prefix* を含む長さ k の時系列パターンの処理で利用できた可能性のある ILP-list が再利用されず, キャッシュミスとなる可能性がある. ここで, 図 1 に対し, 従来手法 PAID により最小サポート値 (回数) 4 とした時の時系列パターン A → B まで処理を行った場合の出力結果を示す (図 6). 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.

次に図 3 と図 4 の SID 3 を対象に図 6 を用いて ILP-list の利用効率とキャッシュミスの原因についての説明を行う. 長さ 2 の時系列パターン A → B を処理した場合, SID 3 の $k-1$-pbp (時系列パターン A の TID) は 2 であり, SID 3 の ILP-list は処理対象と判断できる. 従来手法に従い長さ 3 の時系列パターン A → B → A を処理した場合, SID 3 の $k-1$-pbp (時系列パターン A → B の TID) は NULL であるため, SID 3 の ILP-list に関する処理は行う必要がない. このため, 時系列パターン A → B を処理した時の SID 3 の ILP-list が時系列パターン A → B → A の処理時に CPU キャッシュに存在していても再利用されない. しかし, 長さ k の時系列パターン A → C の処理では時系列パターン A → B と *prefix* が共通しており, *common prefix* であるため, SID 3 の ILP-list は処理対象とな

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, 6
3	A	2, 3, 6
	B	1
	C	5
	D	4
4	A	3, 5
	B	4
	C	1, 2
	D	6
5	A	2 , 4
	B	3
	C	1, 5
	D	6

図 3 Position List of DB

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

図 4 Item Last Position Table

A

Item ID	A	B	C	D
Support	5	4	4	5

-4 -4 -1

A→A

Item ID	A	B	C	D
Support	1	0	3	5

図 5 sequential pattern A and A → A projected DB Supports

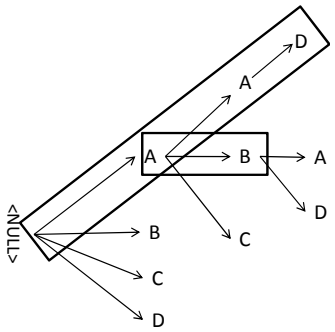

図 6 従来手法の処理パターン

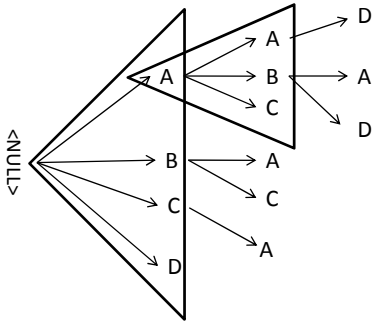

図 7 提案手法の処理パターン

る. また, 長さ 3 の時系列パターン A → B → A の処理が終了した際, 仮に CPU キャッシュから SID 1 の ILP-list が外れてしまった場合を考えると, 時系列パターン A → B → C の処理に移行した際, SID 1 の ILP-list の処理でキャッシュミスが生じる.

このため, 提案手法 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同時に処理する. ここで, 提案手法を用いた例として, 図 6 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に発見されている時系列パターンを示す (図 7). 次に *common prefix* が A である長さ 2 の時系列パターンの処理の説明を, 図 1 の SID 2 を対象として行う. *common prefix* である時系列パターン A の TID は 1 ($k-1$-pbp) であり, これを基に図 3 の SID 2 の position list から時系列パターン 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 と略す) としてまとめる. 例において, SID 2 の multi-k-pbp は multi-k-pbp= $\{A:4,\ B:3,\ C:5\}$ と表わされる. 次に, SID 2 の ILP-list を対象に A → A, A → B, A → C で順に $k-1$-pbp, k-pbp の位置合わせを行い (図 8), 非出現となるアイテムをサポートから差し引く. 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 ですぐにアクセスされるため再利用率が向上し, 時間的局所参照性の改善が可能となる.

4.2 Rebuild-ILP の提案

時系列パターンマイニングでは時系列パターン長が増加していくと, 最小サポート値を満

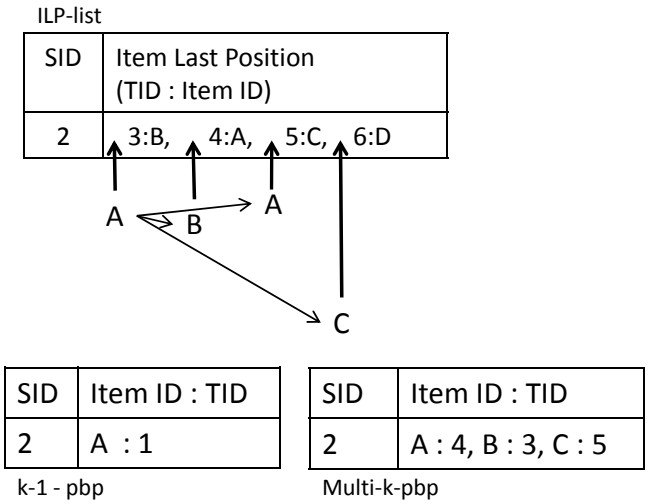

図 8 複数の時系列パターンによる位置合わせ処理

たさないアイテムが増えてくる. このため, 最小サポート値を満たさないアイテムは PAID アルゴリズムにおいて差し引き対象から外すことが可能と考えられる. しかし, 従来手法の PAID では ILP-DB は固定されており, 時系列パターン長が増加して最小サポート値を満たさないアイテムが増えても, それらのアイテムがデータ構造に残ったままである. 例えば, 図 7 より, 長さ 3 の時系列パターン A → B → A を処理した場合, *common prefix* A → B を含む時系列パターンとして時系列パターン A → B → C は存在しない. しかし, SID 1 の ILP-list では Item C が差し引き対象となる. また, 差し引き処理の他にも $k-1$-pbp, k-pbp の位置合わせの処理が行われるため, 最小サポート値を満たさないアイテムを含んだ状態では ILP-list に対して不必要なアクセスが行われる可能性がある.

このため, 提案手法 Rebuild-ILP では *common prefix* を含む長さ k の時系列パターンにより, 最小サポートを満たすアイテムのみで ILP-list の再構築を行う. ここで, 図 4 に対し, 長さ 3 の時系列パターン A → B → A の処理を行う場合の再構築された ILP-DB の例を示す (図 9). 時系列パターン A → B → A の処理では ILP-DB において *common prefix* A → B 以降の範囲で最小サポート値を満たすアイテムは A と D であるため, 図 9 のように処理範囲を削減できる.

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

図 9 再構築された ILP-DB

再構築を行う上で, ILP-list に含まれる最小サポートを満たすアイテムを探索する処理はコストがかかると考えられる. そのため, 提案手法 Common-prefix-at-a-time を応用する. Common-prefix-at-a-time では multi-k-pbp の取得が行われるが, これらは長さ k の時系列パターンのみを対象に行われる. このため, k-pbp を取得する際, 対象としているアイテムの最終出現位置を取得すれば, multi-k-pbp の処理が終了するのと同時に最小サポート値を満たすアイテムのみで ILP-list を再構築できる. 例として, *common prefix* A → B を含む長さ 3 の時系列パターンの処理を行う場合の SID 1 を対象とした再構築処理を示す (図 10). 時系列パターン A → B → A, A → B → D の順で k-pbp を取得する際, 対応するアイテムの最終出現位置の取得を行い ILP-list を構築する.

4.3 マルチコアプロセッサを用いた並列化方式

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と予測し, 本提案の並列化手法として時系列データ単位でのデータ分割を採用する. なお, 従来では時系列パターンに関連したサポートから SID 毎の処理で直接非出現のアイテムの差し引き処理が行われているが, 並列処理では排他制御が必要となり, SID 毎に差し引き処理を行うことはボトルネックとなる. このため, 各スレッド毎で非出現のアイテムの加算を行い, 各スレッドの処理が終えた際に, それらをサポートから差し引く.

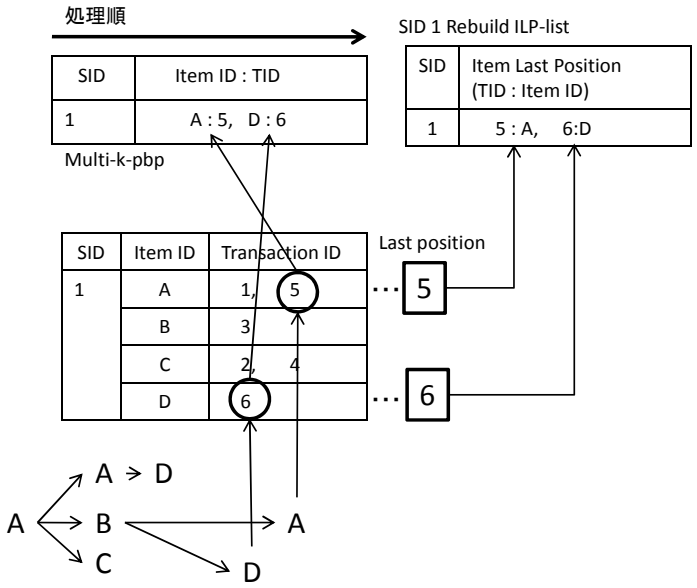

図 10 ILP-list の再構築処理

5. ま と め

PAID アルゴリズムで利用される ILP-list の CPU キャッシュでの利用効率に着目し, 処理対象を共通する prefix を含む複数の時系列パターンとすることにより, ILP-list の再利用率を向上させ, 時間的局所参照性を改良する手法として Common-prefix-at-a-time の提案を行った. また, ILP-list での探索と差し引き対象となる範囲を縮小させ, データへのアクセス回数を減らすために, 共通する prefix における処理で, 最小サポート値を満たすアイテムのみで ILP-list を再構築する手法として Rebuild-ILP の提案を行った. さらに, 並列化手法についても提案を行った.

今後の課題として, 提案手法の実装と性能評価が挙げられる. また, ILP-list の他に position list の処理においてもキャッシュミスが発生する可能性があり, 改良していく必要があると考えられる.

Vol.2010-DBS-151 No.9
2010/11/12

参 考 文 献

1) R. Srikant and R. Agrawal. Mining sequnetial patterns: Generalizations and Performance Improvements. *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, pp.3 - 17(1996).
2) MOHAMMED J. ZAKI, SPADE: An Efficient Algorithm for Mining Frequent Sequences, *Machine Learning*, Vol.40, pp.31-60(2001).
3) Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, Vol.16, No.11, pp.1424 - 1440(2004).
4) Jiahong Wang, Yoshiaki Asanuma, Eiichiro Kodama, Toyoo Takata and Jie Li. Mining Sequential Patterns More Efficiently by Reducing the Cost of Scanning Sequence Databases, *IPSJ Journal*, pp.3365 - 3379(2006).
5) Zhenglu Yang, Yitong Wang and Masaru Kitsuregawa. Effective Algorithms for Sequential Pattern Mining, 日本データベース学会 Letters, Vol.5, No.1, pp.53-56(2006).
6) Zhenglu Yang, Yitong Wang and Masaru Kitsuregawa. PAID: Mining Sequential Patterns by Passes Item Deduction in Large Databases, *Proceedings of the Tenth International Database Engineering and Applications Symposium*, pp.113-120(2006).
7) Mohammed J. Zaki, Parallel Sequence Mining on Shared-Memory Machines. *Journal of Parallel and Distributed Computing*, Vol.61, pp.401-426(2001).
8) 高木 充, 田村 慶一, 周藤 俊秀, 北上 始. 分散型ワーカーモデルによる Modified PrefixSpan 法の並列処理とその動的負荷分散手法, 第 16 回データ工学ワークショップ, (2005).
9) Jiawei Han, Jian Pei, Yiwen Yin, Mining frequent patterns without candidate generation, *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp.1 - 12(200).
10) Amol Ghoting, et al., Cache-consious Frequent Pattern Mining on a Modern Processor, *Proceedings of the 31st VLDB Conference*, pp.577-588(2005).
11) Wm. A. Wulf, Sally A. McKee, Hitting the Memory Wall: Implications of the Obvious, *ACM SIGARCH Computer Architecture News*, pp.20-24(1995).
12) Jay Ayres, Johannes Gehrke, Tomi Yiu, Jason Flannick, Sequential Pattern Mining using A bitmap Representation, *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.429-435(2002)