CCDR-PAID: More Efficient Cache-Conscious PAID Algorithm by Data Reconstruction

Yuki Matsubara, Jun Miyazaki, Goshiro Yamamoto, Yuki Uranishi, Sei Ikeda and Hirokazu Kato
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma
Nara, Japan
{yuki-m, miyazaki, goshiro, uranishi, sei-i, kato}@is.naist.jp

ABSTRACT

In this paper, we propose the CCDR-PAID algorithm which is a technique to improve CPU cache utilization of sequential pattern mining more efficiently as an extension of existing CC-PAID which is our previous work. Compared to PAID, CC-PAID improves temporal locality by changing the access pattern to data structures and processing multiple sequential patterns with a common prefix at a time to reduce the memory access latency by suppressing CPU cache misses. In this paper, we extend CC-PAID and dynamically reconstruct data structures, so that unnecessary data access to them can be avoided. The experimental results showed that CCDR-PAID executes up to 25% faster than CC-PAID because it sufficiently reduces cache misses and, therefore, provides better CPU cache utilization due to compaction of the data structure.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—Data mining

General Terms

Algorithms, Performance, Measurement, Experimentation

Keywords

Sequential pattern mining, CPU cache utilization, Performance evaluation

1. INTRODUCTION

Sequential pattern mining is a technique to find the subsequences that satisfy a certain threshold while maintaining the order of their appearance, called the sequential patterns, from a database consisting of sequence data. This is an important technique in many research fields, such as data analysis, behavior prediction, information recommendation, etc. In the research field of the sequential pattern mining, processing time tends to greatly increase when a large database and a small threshold are used. The processing time is still regarded as one of the most important issues to be solved. One solution is to use parallelism in sequential pattern mining algorithms. In the last decade, many parallel algorithms have been proposed.

Sequential pattern mining algorithms tend to perform recursive access to specific data structures. If the total amount of accesses to the data structures exceeds the size of the CPU cache, it causes long latency to refer to the data on main memory due to cache misses. Though multi-core CPUs have made remarkable progress in recent years, the speed of the main memory cannot catch up with that of CPUs. In other words, the memory wall [7] problem has not been solved yet. Since the access latency caused by CPU cache misses becomes a bottleneck in the process of sequential pattern mining, a cache-conscious algorithm, called CC-PAID which is our previous work [4], which is an extension of the PAID algorithm [8], has been proposed to reduce the CPU cache misses.

CC-PAID aims to reduce execution time by augmenting CPU cache utilization which is mainly achieved by improving temporal locality for accessing data structures. In the PAID algorithm, the sequential patterns with a common prefix sequence can share a range of specific data structures to be processed. CC-PAID improves the temporal locality of such data structures by changing the access pattern to the data, for example, by processing similar sequential patterns at a time. The evaluation of CC-PAID confirmed that cache misses can be reduced by around 60% and the processing speed becomes around twice faster than PAID. According to these results, reducing the processing time by improving CPU cache utilization is very effective for sequential pattern mining.

In this paper, we propose the CCDR-PAID algorithm, an extension of CC-PAID, aiming to improve the CPU cache utilization more effectively than CC-PAID, and evaluate it. CCDR-PAID dynamically reconstructs the specific data structure which is used in two processing steps, so that it can improve the efficiency of accessing to the data structure. The reconstruction mainly deletes unnecessary data which have not been used any longer. This idea brings (1) efficient CPU cache utilization by avoiding the reference to specific data structure, and (2) reducing cache misses due to the compaction of specific data structure.

2. PRELIMINARY

For the sake of readability, we explain some terminologies used in this paper. Sequential pattern mining is to discover all the possible sequential patterns that are subsequences in a sequence database (SDB), while satisfying a given mini-

SID \ TID	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

Figure 1: A 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

Figure 2: Sequential patterns where minimum support is four.

mum support. A k-subsequnece is a subsequnece of length k. Similarly, a k-sequential pattern is a sequential pattern of length k.

A sequence database (SDB) is constructed by multiple sequence data, and a sequence data consists of itemsets. An SDB is represented as $SDB = < s_1, s_2, \cdots, s_n >$, where s_k is a sequence data with a sequence ID (SID). It is usually arranged in order of SID. Sequence s is represented as $s = < is_1, is_2, \cdots, is_m >$, where is_k is an itemset with a transaction ID (TID) which indicates time or order of the items occurrence. The itemsets are also arranged in order of TID. The itemset is is represented as $is = (i_1, i_2, \cdots, i_c)$, where i_k is an item. Let k-subsequence α be a combination of the items keeping the order relation by TID in a sequence. A support is defined as the ratio or the number of occurrences of α to that of all sequence data. A threshold support value given in advance by the user is called a minimum support.

In the discovery process of sequential patterns, a sequential pattern mining algorithm counts supports of subsequences by scanning an SDB, and then, all k-subsequences which satisfy a minimum support are discovered as sequential patterns. The process of sequential pattern mining is divided into an itemset-extention step (I-Step) which derives the combinations of the items satisfying a minimum support in an itemset, and a sequence-extension step (S-Step) which generates sequential patterns [1]. We show an example of an output as a result of sequential pattern mining. Figures 1 and 2 show a SDB containing the sequence data which consist of the itemsets of length 1 as an example of finding sequential patterns, where the minimum support is four. In Fig. 2, the notation $A \rightarrow B \rightarrow A : 4$ means that subsequence $A \rightarrow B \rightarrow A$ of length 3 occurs 4 times in the SDB in Fig. 1.

3. RELATED WORK

This section introduces some related studies on sequential pattern mining and CPU cache utilization of frequent pattern mining.

Pei et al. proposed the PrefixSpan algorithm [5] that discovers $k+1$-sequential patterns satisfying a minimum support in an SDB by using k-sequential patterns found in the previous step. The search space to find the items which follow the k-sequential patterns in a SDB is called a projected sequence database. PrefixSpan discovers the items which satisfy a minimum support in a projected sequence database, and then, $k+1$-sequential patterns are derived by appending these items to the k-sequential patterns. Since the search space can be narrowed as the length of sequential patterns becomes longer, PrefixSpan can discover new sequential patterns more effectively.

Wang et al. proposed the FSPM algorithm [6] that improves the performance by reducing the cost of scanning a SDB. They noticed that PrefixSpan scans the same items in a SDB in each discovering process of sequential patterns. For each item that satisfies a minimum support, FSPM divides the discovery process for k+1-sequential patterns. It chooses one item that satisfies a minimum support from them, and discovers all sequential patterns including the item. FSPM can reduce the search space in an SDB because the unnecessary items used in the previous step are properly removed from it.

The PAID algorithm [8] proposed by Yang et al. effectively makes use of item-last-position lists which store the positions of the last occurrence of an item satisfying a minimum support in an sequence data and the corresponding support counts of k-sequential patterns. PAID looks up the items which do not appear after the positions of a k-sequential pattern in a sequence data by using the item-last-position list, though these items appear after the position of a $k-1$-sequential pattern. Since these items do not occur any more after the position of a k-sequential pattern, their support counts are deducted from the support of the k-sequential pattern obtained in the previous step. This reduces the overhead of updating support counts.

On the other hand, in the area of association rule mining, Ghoting et al. proposed the cache conscious algorithm [2] which improves CPU cache utilization as an extension of the FP-growth algorithm [3]. They improved the spatial locality of a data structure in FP-growth by relocating the data, so that they can be accessed sequentially by gathering correlated data in one place. It also divides the data into the tiles, each of which fits into a cache of CPU, in order to improve the temporal locality and to reduce data fetch time. Moreover, it improves the ratio of reusing the same data by multi-threading to achieve further speedups.

4. THE CC-PAID ALGORITHM

The CC-PAID algorithm which is our previous work is a cache conscious version of PAID improves the temporal locality of the data structures of PAID, so that CPU cache utilization can increase, which leads to better execution time [4].

CC-PAID achieved more efficient CPU cache utilization by changing the data access pattern during the steps of the algorithm without modifying the core of the PAID algorithm. Sequential patterns can be represented as a tree structure [1]. The PAID algorithm selects a k-sequential pattern for discovering $k+1$-sequential patterns one by one by means of traversing the tree in a depth-first manner. In contrast, CC-PAID processes k-sequential patterns which include $k-1$-common prefix at a time. It traverses the tree in a breadth-first manner when finding child nodes. This strategy makes use of CPU cache. Once the data structure related to the calculation of support counts is fetched to the cache when traversing a child node in the tree, the data is immediately reused for processing its sibling nodes, which reduces the number of cache misses. Figures 3 and 4 show the examples of access patterns of PAID and CC-PAID, when sequential pattern $A \rightarrow B \rightarrow A$ in Fig. 1 is discovered. The sequential patterns in a box indicate that all of them have already been processed. When CC-PAID discovers sequential pattern $A \rightarrow B \rightarrow A$ in this example, it processes 2-sequential patterns $A \rightarrow A$, $A \rightarrow B$, and $A \rightarrow C$ whose common prefix pattern is a 1-sequential pattern A in a breadth-first manner.

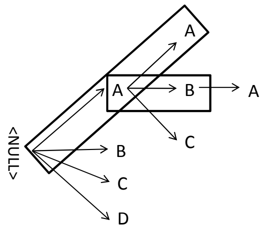


Figure 3: Access pattern of PAID.

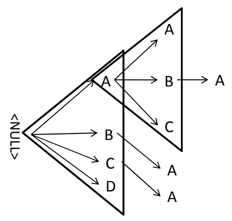


Figure 4: Access pattern of CC-PAID.

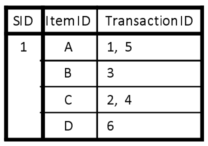
CC-PAID accesses two data structures for each sequence data. One is the position lists which store transaction IDs grouped by each item type in a sequence data. The other is the item-last-position lists(ILP-lists), which consist of a list of elements that are pairs of a 1-sequential pattern and a TID which represents the last position of the item in a sequence data.

In order to calculate support counts for each sequence data. First, CC-PAID determines the TIDs in which k-sequential patterns appear in a sequence data. Let a $k-1$-prefix border position ($k-1$-pbp) be the TID of the $k-1$-common prefix in k-sequential patterns. CC-PAID finds k-pbps larger than a $k-1$-pbp from position lists. This process is performed for k-sequential patterns. The obtained k-pbps are recorded along with k-sequential patterns as a multi-k-pbp.

Next, for each k-pbp in the multi-k-pbp, it looks for an element position which including TID which is larger than the k-pbp in an ILP-list, and deducts support counts by using the ILP-list. The position of k-pbp in an ILP-list is called E_k-position. When searching the element in the ILP-list, we explore the position of elements which is larger than k-pbps and is after the element associated with S_{k-1}-position. Since searching positions in the ILP-list is made in one way, An E_k-position can be reused as a S_{k-1}-position in an ILP-list in the next step to discover $k+1$-sequential patterns. Note that the items in the elements between S_{k-1}-position and E_k-position are ensured not to occur as the $k+1$-th items in the $k+1$-subsequences in a sequence data. Therefore, the counts of these items can be deducted from their support counts in a database of projected ILP-lists related to $k-1$-sequential patterns, which is a set of the elements which appear after S_{k-1}-positions in all ILP-lists for k-sequential patterns. These support counts are organized as a $k-1$-support list.

In the process of discovering $k+1$-sequential patters by using k-sequential patterns which have a $k-1$-common prefix, CC-PAID manages the position lists and the ILP-list corresponding to sequence data to be processed at a time. When this process finishes, the $k+1$-sequential patterns satisfying a minimum support are discovered because the support counts of $k+1$-subsequences can be calculated with $k-1$-support lists corresponding to k-sequential patterns.

We show an example of processing sequential patterns $A \rightarrow A$, $A \rightarrow B$, and $A \rightarrow C$ where the minimum support is 4, by using the data with SID $= 1$ in the sequence database in Fig. 1. First, we obtain a multi-k-pbp in the position lists of items A, B and C as shown in Fig. 5, which corresponds on the data with SID $= 1$ in Fig. 1. Since the $k-1$-pbp is 1, we look for a TID which is larger than



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

Figure 5: Position list table of SID:1 in Fig 1.

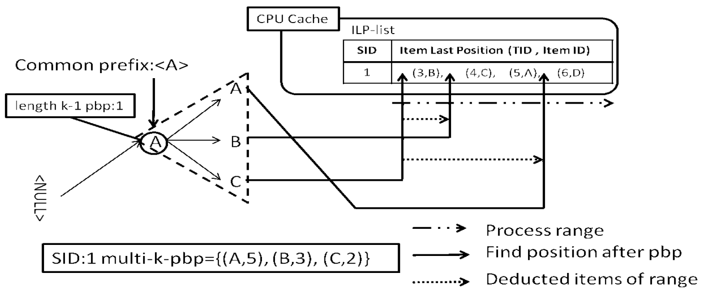


Figure 6: Process of an ILP-list of SID:1 in Fig 1.

1 for each k-sequential pattern. The obtained k-pbps are stored along with the k-th item of a k-sequential pattern. At this point, a multi-k-pbp becomes $\{(A,5),(B,3),(C,2)\}$. Next, we find the elements including the TIDs which are larger than k-pbps from the sub-list after S_{k-1}-position in the ILP-list in Fig. 6. The support counts of the items in these elements between S_{k-1}-position and E_k-position are deducted. Fig. 6 shows when sequential patterns $A \rightarrow A$, $A \rightarrow B$, and $A \rightarrow C$ are processed. Focusing on sequential pattern $A \rightarrow A$, the items A, B, and C, have their support counts deducted because k-pbp is 5. The support counts of these items are deducted from the support list corresponding to sequential pattern A. The support list of sequential pattern A becomes support list a sequential pattern $A \rightarrow A$ when all sequence data are processed. For other sequential patterns $A \rightarrow B$ and $A \rightarrow C$, the same process is applied.

CC-PAID processes k-sequential patterns which includes a $k-1$-common prefix at a time. An ILP-list fetched to CPU cache can immediately be reused by the processes to find other $k+1$-sequential patterns. Hence, the temporal locality, i.e., CPU cache utilization, of the ILP-list is improved. In addition, this strategy of CC-PAID can also improve the CPU cache utilization of other data structures, such as position lists. The evaluation of CC-PAID showed that CC-PAID has about 2x faster execution time and about 60% less CPU cache misses than PAID.

5. THE CCDR-PAID ALGORITHM

In this paper, we propose a cache-conscious PAID with data reconstruction, named CCDR-PAID, which is a more efficient data access method for CC-PAID to obtain further speedups of sequential pattern mining. To reduce execution time, CC-PAID improves CPU cache utilization by changing access pattern to the data structures corresponding to discovered sequential patterns. As a result, the improvement of CPU cache utilization is important and effective for sequential pattern mining. We consider another approach to augment CPU cache utilization other than the improvement of data access patterns as CC-PAID adopts. Sequential pattern mining tends to cause recursive access to specific data structures, e.g., position lists and ILP-lists in CC-PAID. Be-

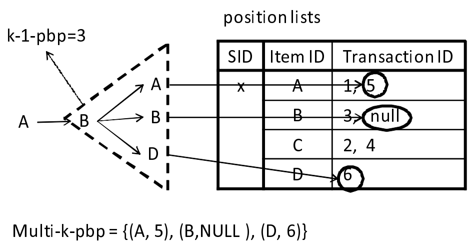


Figure 7: Access to position lists in the CC-PAID algorithm.

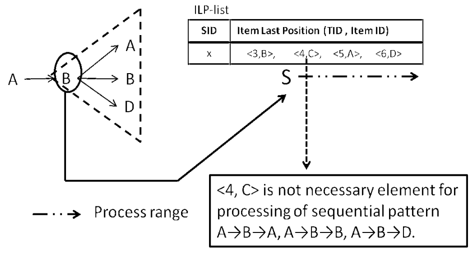

ILP-list
SID
Item Last Position (TID , Item ID)
x
<3,B>, <4,C>, <5,A>, <6,D>
A
B
A
B
D
S
Process range
<4, C> is not necessary element for processing of sequential pattern A→B→A, A→B→B, A→B→D.

Figure 8: Process range in the CC-PAID algorithm.

cause the way to access to data structures affects CPU cache utilization, we focus on the data structures of position lists and ILP-lists in CC-PAID.

The processing flow and its corresponding data structures in CC-PAID and CCDR-PAID are showed in Fig. 9. The processing in CC-PAID is divided into two parts: one is finding multi-k-pbps, and the other is updating support counts for each k-sequential pattern and sequence data. On finding a multi-k-pbp, position lists need to be accessed when each k-sequential pattern is processed. The updating support counts consists of finding items of which support counts need to be deducted in an ILP-list and calculating their new support counts, which need to be processed for each k-sequential pattern. In addition, the deduction of support counts also requires ILP-lists.

5.1 Data Access to Position Lists

A position list is used to find a k-pbp of a k-sequential pattern. No k-sequential pattern in a sequence data to be processed might be found when obtaining a k-pbp, even if a k-sequential pattern exists in a sequence database. In this case, the k-pbp becomes null in the sequence data. For example, Fig. 7 shows that the position lists are being processed after 3-sequential patterns $A \rightarrow B \rightarrow A$, $A \rightarrow B \rightarrow B$, and $A \rightarrow B \rightarrow D$ are discovered in a sequential database. During processing 3-sequential pattern $A \rightarrow B \rightarrow B$, the position list of item B is scanned. As a result, the obtained k-pbp is found to be null. Since the sequential pattern $A \rightarrow B \rightarrow B$ does not exist in the sequence data.

Consider the same sequential patterns and sequence database as in Fig. 7. If we can determine that no k-sequential pattern exists in a sequence data before finding a k-pbp, the processing related to the k-pbp can be omitted because the k-pbp must be null.

We can easily determine whether a k-sequential pattern exists or not by using an ILP-list.

Since the ILP-list records pairs of an item which is a 1-sequential pattern and the last position that it appears, it is found that the items before S_{k-1}-position in the ILP-list never becomes as the k-th ones in k-sequential patterns in a sequence data. Therefore, we can skip to find k-pbps, if we check whether the k-th items in k-sequential patterns exists after S_{k-1}-position or not.

5.2 Data Access to ILP-lists

An ILP-list is used to examine whether the support counts of items need to be deducted or not. In CC-PAID, there is a case in which the elements in ILP-lists to be processed may include the items which cannot become the $k+1$-th ones of $k+1$-sequential patterns. We can determine it depending on whether the items exist as the k-th ones of k-sequential patterns or not. This is because these items are not satisfied with a minimum support unless they are the k-th ones. For example, the possible the 4-th items of 4-sequential patterns are A, B, and D, because 3-sequential patterns $A \rightarrow B \rightarrow A$, $A \rightarrow B \rightarrow B$, and $A \rightarrow B \rightarrow D$ exist as shown in Fig. 8. However, item C is determined to be useless because it is included after S_{k-1}-position.

The FSPM algorithm classifies items into two sets. One includes the items which can become the $k+1$-th ones as a candidate-set and the other has the rest of them. FSPM processes only a candidate-set for updating support counts, called candidate-driven manner. The same technique of a candidate set can be applied to an ILP-list.

Based on above discussions, if ILP-lists contain only necessary data to be processed, it is possible to access efficiently. Therefore, we propose CCDR-PAID which dynamically reconstructs ILP-lists to keep them compact during execution.

5.3 Reconstruction of ILP-list

When obtaining a multi-k-pbp with position lists, we begin to scan an ILP-list from S_{k-1}-position and check whether the k-th items of k-sequential patterns exist or not, so that we can determine k-sequential patterns which do not need to be processed. Updating support counts also requires ILP-lists. The items of the k-sequential patterns appeared between S_{k-1}-positions and E_k-positions in the ILP-list are looked up and their support counts are deducted.

It is important that these two processes share the same data, i.e., ILP-lists. Therefore, if only the necessary data in an ILP-list is extracted and reconstructed to a compact ILP-list before these processes, unnecessary access to the ILP-list can be avoided by using a compact ILP-list. If both these processes access the required data independently and reconstruct the ILP-list, the processing cost increases. However, if the reconstruction of an ILP-list and accesses to it are shared by them, it becomes advantageous in terms of the cost. In addition, reconstructed ILP-lists are shared and reused by other discovering processes for k-sequential patterns in each sequence data in CCDR-PAID.

The reconstruction of an ILP-list is done by replicating the required data in an old ILP-list to a new one in CCDR-PAID, in order to avoid the indirect memory access, i.e., dereference of pointers, to the new ILP-list. Considering CPU cache, the reconstruction of an ILP-list reduces cache misses due to the compaction of the data, while it consumes more memory space because of the data replication.

5.4 Process Flow of CCDR-PAID

We explain the process flow in CCDR-PAID until finishing the update of support counts by using examples and Fig. 9.

First, we choose the ILP-list currently being used. We extract and replicate all the items that are the k-th ones in

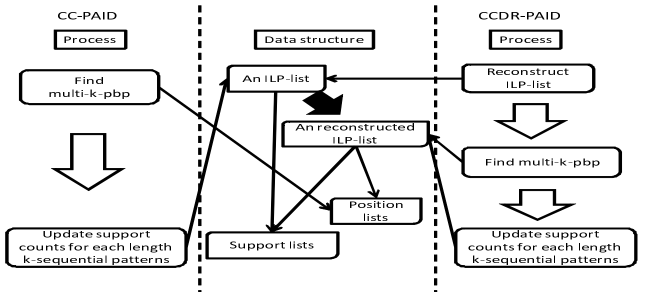

CC-PAID
Process
Find multi-k-pbp
Update support counts for each length k-sequential patterns
Data structure
An ILP-list
An reconstructed ILP-list
Position lists
Support lists
CCDR-PAID
Process
Reconstruct ILP-list
Find multi-k-pbp
Update support counts for each length k-sequential patterns

Figure 9: Processes and data structures in CC-PAID and CCDR-PAID.

k-sequential patterns located after S_{k-1}-position in the ILP-list and their corresponding TIDs. A reconstructed ILP-list is created with the replicated items and TIDs.

For example, consider the 3-sequential patterns $A \rightarrow B \rightarrow A$, $A \rightarrow B \rightarrow B$, and $A \rightarrow B \rightarrow D$ as shown in Fig. 10. When TID of the common prefix $A \rightarrow B$ is 3, we start to look up the ILP-list to be used from item C and find items A, B, and D. As a result, pairs of an item and a TID, which are last positions, <5, A> and <6, D>, are extracted. These data become the elements of a reconstructed ILP-list.

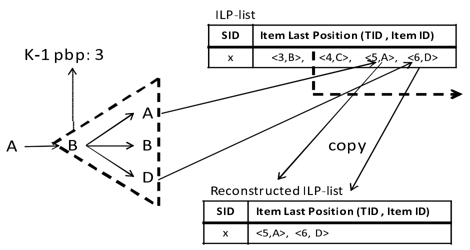
Next, we access the reconstructed ILP-list, so that we can obtain a multi-k-pbp by looking up the position list associated with the items in the reconstructed ILP-list. Items A and D are, for example, included in the reconstructed ILP-list shown in Fig. 11. We obtain each k-pbp by accessing the position lists associated with them. CCDR-PAID can skip to look up the position list of item B, though existing approaches need to process it.

Lastly, we look up the items of which support counts need to be deducted for k-sequential patterns by accessing the reconstructed ILP-list again, and subtract their support counts. The reconstructed ILP-list can also be reused for the next step to discover $k+1$-sequential patterns. For example, the data required to be processed in the reconstructed ILP-list are <5, A> and <6, D> in Fig. 12. Therefore, CCDR-PAID can reduce the number of the data to be processed, because existing methods also need to process <4,C> as well as <5, A> and <6, D>.

6. EXPERIMENTAL RESULTS

In this section, we evaluate and compare our CCDR-PAID with existing the PAID and CC-PAID algorithms. We used three different types of data sets which were created by the IBM data generator. Table 2 shows the parameters used for generating the data sets. The evaluation was conducted under the environment as shown in Table 1. The criteria of the evaluation are execution time and the number of cache misses occurred at L3 cache. The L3 cache misses were measured by Intel VTune Amplifier XE 2011. Note that we focused only on S-step for the evaluation, because I-step is not an essential part of sequential pattern mining. The minimum support values in graphs are indicated in ratio.

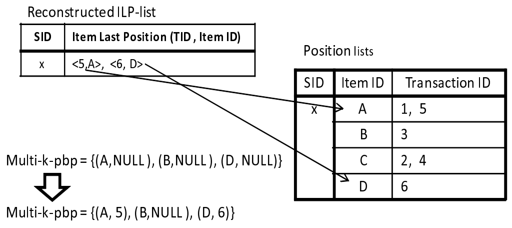
- Dataset D_1 with $0.24 \sim 0.3$ of minimum supports (Fig. 13)
  The execution time of CCDR-PAID where the minimum support is 0.24 reduces by around 16%, and the number of L3 cache misses where the minimum support is 0.26 also decreases by around 18% compared to CC-PAID.



SID	Item Last Position (TID , Item ID)
x	<3,B>, <4,C>, <5,A>, <6,D>

SID	Item Last Position (TID , Item ID)
x	<5,A>, <6, D>

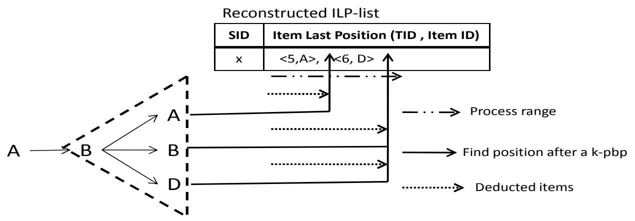
Figure 10: Reconstruction of ILP-list.


Reconstructed ILP-list
Position lists
Multi-k-pbp = {(A,NULL), (B,NULL), (D, NULL)}
Multi-k-pbp = {(A, 5), (B,NULL), (D, 6)}

SID	Item Last Position (TID , Item ID)
x	<5,A>, <6, D>

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

Figure 11: Finding multi-k-pbp by using a reconstructed ILP-list.



SID	Item Last Position (TID , Item ID)
x	<5,A>, <6, D>

Figure 12: Update of support counts using a reconstructed ILP-list.

- Dataset D_2 with $0.6 \sim 0.66$ of minimum supports (Fig. 14)
  The execution time of CCDR-PAID where the minimum support is 0.6 shortens by around 10% compared to CC-PAID. The number of L3 cache misses drops down by around 16% compared to CC-PAID.
- Dataset D_3 with $0.08 \sim 0.14$ of minimum supports (Fig. 15)
  CCDR-PAID where the minimum support is 0.08 runs around 25% faster and has around 28% less L3 cache misses than CC-PAID.

These measurements revealed that CCDR-PAID runs faster and the number of L3 cache misses reduces compared to both PAID and CC-PAID for all combinations of data sets and minimum supports that we used. In particular, the number of L3 cache misses decreases as that of kinds of items increases. The reason is that most subsequences do not satisfy when sequential patterns of short length are being generated, which leads to a drastic decrease in the sizes of ILP-lists and accesses of position lists. Therefore, the number of the data to be referenced in these lists decreases in the early time of execution.

7. CONCLUSIONS

In this paper, we proposed a more efficient data access method for a data structure in CC-PAID, called CCDR-PAID, to enable sequential pattern mining to execute faster.

OS	Fedora 13 64bit	
Kernel	2.6.34.7-61	
CPU	Intel Core i7 980X	
	Frequency	3.33GHz
	Core number	6
	L2 cache	256KB x 6
	L3 cache	12MB
Main memory	12GB	

Table 1: Experimental environment.

	Data set name		
	D_1	D_2	D_3
Number of sequences	50000	50000	50000
Average transactions per sequence	50	100	50
Average items per transaction	4	4	4
Number of different items	350	350	700

Table 2: Data set parameters.

CCDR-PAID improves CPU cache utilization by using reconstructed ILP-lists, which reduces the number of accesses to position lists in the process of finding some data, e.g., multi-k-pbp. In addition, it also reduces the amount of data to be fetched into the CPU cache in updating support counts. The results showed that CCDR-PAID can perform up to 25% faster and have up to 28% less cache misses than the original CC-PAID for all data sets and minimum supports that we used in the experiments. In the future, we intend to use many-core processors such as a GPGPU to pursue more efficient execution of sequential pattern mining.

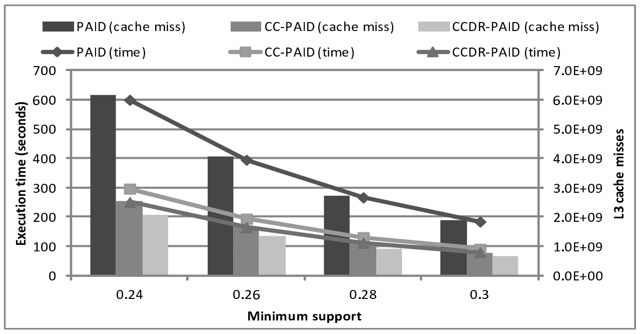


Figure 13: Execution time and L3 cache misses using D_1.

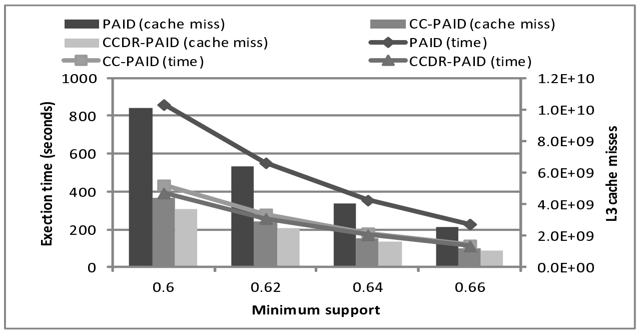


Figure 14: Execution time and L3 cache misses using D_2.

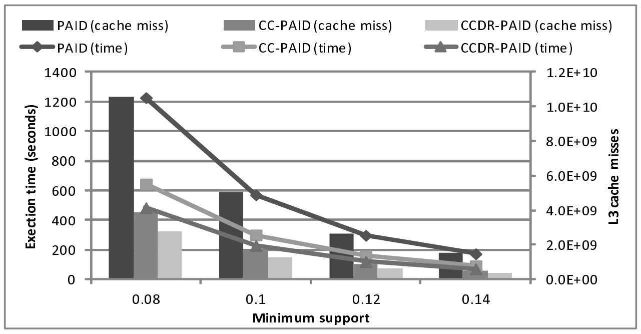


Figure 15: Execution time and L3 cache misses using D_3.

Acknowledgments

This work was supported in part by MEXT (Grant-in-Aid for Scientific Research (A) #22240005 and Grant-in-Aid for Scientific Research (C) #23500121).

8. REFERENCES

[1] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '02, pages 429–435, New York, NY, USA, 2002. ACM.

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

[3] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.

[4] Y. Matsubara, J. Miyazaki, M. Fujisawa, T. Amano, and H. Kato. Cc-paid: A cache-conscious parallel sequential pattern mining algorithm. IPSJ Transactions on Databases, 4(2):88–100, Jul 2011(in Japanese).

[5] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering, 16:1424–1440, 2004.

[6] J. Wang, Y. Asanuma, E. Kodama, T. Takata, and J. Li. Mining sequential patterns more efficiently by reducing the cost of scanning sequence databases. IPSJ Digital Courier, 2:768–782, 2006.

[7] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News, 23:20–24, March 1995.

[8] Z. Yang, M. Kitsuregawa, and Y. Wang. Paid: Mining sequential patterns by passed item deduction in large databases. In Proceedings of the 10th International Database Engineering and Applications Symposium, pages 113–120, Washington, DC, USA, 2006. IEEE Computer Society.