Parallel frequent itemset mining using systolic arrays

Mohammad Karim Sohrabi, Ahmad Abdollahzadeh Barforoush

Abstract

Since extraction of frequent itemsets from a transaction database is crucial to several data mining tasks such as association rule generation, so frequent itemset mining is one of the most important concepts in data mining. One of the major problems in frequent itemset mining is the expansion of the number of results which is directly effecting on the execution time of itemset mining algorithms. To address this problem, closed itemsets have been proposed, which provides concise lossless representations of the original collection of frequent itemsets. Henceforth, the frequencies of all itemsets in the original collection can be reconstructed from the reduced collection. However, the reduction provided by this exact method is not sufficient to solve the pattern explosion problem, mainly because of high dimensional datasets which have large number of items in each transaction. Colossal itemset mining is another solution to reduce the output size which will not be useful if the set of all frequent itemsets have been required. Higher level of performance improvement can be obtained from efficient scalable parallel mining methods. In this paper we represent an efficient scalable parallel algorithm using systolic arrays to conduct mining of frequent itemsets in very large, such as high dimensional, datasets. In our algorithm, we use a bit matrix to compress the dataset and mapping the mining algorithm on the systolic arrays architecture. For this purpose, each transaction of dataset represents as a row in the bit matrix. We use this bit matrix structure to model the pattern mining as a systolic array problem. Our experimental results and performance study show that this algorithm outperforms substantially the best previously developed parallel algorithms.

1. Introduction

One of the important problems in data mining is discovering frequent patterns from transaction databases, where in each transaction contains a set of items. Extraction of frequent patterns plays an essential role in association rules mining process which is used in numerous applications, including analysis of customer purchase patterns, analysis of web access patterns, the investigation of scientific or medical processes, and the analysis of DNA sequences. In this section, we first indicate our problem’s scope and then we present some preliminaries and represent a bit matrix structure which is used for constructing the parallel method of this paper.

1.1. Scope of the problem

There exist several types of frequent pattern mining. The main types of which include: frequent itemset mining, sequential pattern mining, and graph mining. In this paper we concentrate on the itemsets and for simplicity, we use the terms “pattern” and “itemset” interchangeably.

Many efficient frequent pattern mining algorithms have been proposed in the literature [2–9]. Apriori-like algorithms, which are the most important level order mining algorithms, suffer from two main problems: huge numbers of candidate sets, and repeatedly scanning the dataset to check the frequency of each of candidates [2–4]. FP-growth and its successors [5–8] are known as the second generation of pattern mining algorithms which mine frequent patterns of datasets using a depth first search method without candidate generation and without multiple scans on the dataset. High dimensional datasets, which have small number of transactions and large number of items in each transaction, cause creation of vertical algorithms which operate on vertical datasets [9]. In vertical data format, each row of dataset has an item and a set of transaction ids which are containing this item.

Major challenge in mining frequent patterns is the fact that all the sub-patterns of a frequent pattern are frequent and there are an exponential number of sub-patterns for each pattern which generate a huge number of frequent patterns as result. To overcome this problem, closed frequent pattern mining [10–15] was proposed. Closed pattern set is a compact set of frequent patterns which retains all information of complete pattern set. A pattern is considered a closed pattern if it is frequent in the dataset and there exist no super-pattern of it which has the same support as it in the dataset.
Although closed pattern mining has incredibly reduced the amount of computation and the output volume of the mining process, the pattern mining problem is still time and memory space consuming for high dimensional datasets. Colossal pattern mining is a good alternative solution for pattern explosion problem [1,16,17]. The main idea of colossal patterns mining is to find mining methods which extract only large-sized patterns without mining the small-sized and mid-sized patterns. This good idea is suitable for the cases which do not need all frequent patterns and only large-sized patterns are useful for them.

Higher level of performance improvement can be expected from parallel execution of pattern mining algorithms. Parallelism is a good solution to find all (closed) frequent patterns of a large (high dimensional) database which is studied in this paper.

1.2. Preliminaries and definitions

Let \( I = \{i_1, i_2, \ldots, i_n\} \) be a set of items (also called columns). The dataset \( D \) consists of a set of rows (also called transactions) \( R = \{r_1, r_2, \ldots, r_m\} \), where each row \( r_i \) is a set of items with an id called rid or tid. Given a set of items \( X \subseteq I \) we define the support set, denoted \( D(X) \subseteq R \) as the maximal set of rows that contain \( X \). The number of rows in the dataset that contains \( X \) is called the support of \( X \). By definition the support of \( X \) is given as \( \text{Sup}(X) = |D(X)| \).

An itemset \( X \) is called frequent itemset or frequent pattern if \( \text{Sup}(X) \geq \text{minsup} \), where \( \text{minsup} \) is a user specified lower support threshold. In other words, a pattern is frequent if the number of transactions which contains that pattern is not less than the user specified threshold.

A set of items \( X \subseteq I \) is called a closed pattern if there exists no \( Y \) such that \( Y \subseteq X \) and \( \text{Sup}(X) = \text{Sup}(Y) \), i.e., there is no superset of \( X \) with same support. Put another way, the row set that contains the superset \( Y \), must not be exactly the same as the row set that contains the set \( X \). An itemset \( X \) is called a frequent closed pattern, if it is closed and frequent.

Fig. 1 shows an example of a dataset in which the items are represented using alphabets.

The set of items which used in this database is \( I = \{a, b, c, d, e, f, h, k, m, p\} \). In this dataset, \( X = \{b, e, f\} \) is a frequent itemset with support 2 because it is a subset of two transactions 6 and 7. For simplicity, we show this itemset as \( X = bef \). "bef" is not a closed itemset because there exists the super itemset "bcdef" in the dataset with the same support. However "bcdef" is a closed itemset because there exists only one super itemset of it in dataset "bcdef" which have support 1.

Given a dataset \( D \) and a user support threshold \( \text{minsup} \), frequent pattern mining problem is to discover all frequent patterns with respect to \( \text{minsup} \) and similarly frequent closed pattern mining problem is to find frequent closed patterns with respect to \( \text{minsup} \).

We can represent a dataset as a bit matrix [1] with \( m \) rows and \( n \) columns such that each row is corresponding to a transaction of dataset and each column is corresponding to an item of set \( I \). Each row of dataset will be an \( n \)-bit bit string such that if a row of dataset is containing item \( i \) then \( i \)th bit of the correspond bit string set to 1 and else set to 0. Therefore our data set is compressed in a bit matrix. For example the bit matrix corresponding to dataset of Fig. 1 is presented in Fig. 2.

Two arrays were constructed with bit matrix: colSum is an \( n \)-member array that saved the frequency (support) of each item, and rowSum is an \( m \)-member array which saved the number of items in each row. We can perform a column wise pruning on the bit matrix based on \( \text{minsup} \) and eliminate all the columns which their corresponding sum in colSum is less than \( \text{minsup} \). Third column of Fig. 1 shows the pruned dataset based on threshold \( \text{minsup} = 2 \). For this threshold the bit matrix is pruned by eliminating columns which their corresponding colSum is less than 2 as be shown in Fig. 3.

To determine whether an itemset \( X \) is frequent or not, we construct its colAndVector by operating operator "AND" on the columns corresponding to all the items of \( X \). If the sum of values in a colAndVector is not less than \( \text{minsup} \) then the itemset of that colAndVector is frequent. For example, colAndVector\((ab)\) has only one bit with value 1 (third bit) and so \( ab \) is not frequent.

To indicate closeness of a frequent itemset such \( X \) from the bit matrix, we should construct the colAndVector of \( X \) and operate AND on colAndvector with column of each item that does not exist in \( X \) separately. If the result of at least one of these AND operations is equal with colAndVector of \( X \) then \( X \) is not closed. For example, we can see in Fig. 3, colAndVector\((bef)\) = 00000110 and colAndVector\((bdef)\) = 00000110 and so \( bef \) is not a closed pattern.

<table>
<thead>
<tr>
<th>Tid</th>
<th>Itemset</th>
<th>Frequent Items (minsup=2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, c, f, p</td>
<td>a, c, f</td>
</tr>
<tr>
<td>2</td>
<td>d, e, f, m</td>
<td>d, e, f</td>
</tr>
<tr>
<td>3</td>
<td>a, b, d, f</td>
<td>a, b, d, f</td>
</tr>
<tr>
<td>4</td>
<td>a, c, e, h</td>
<td>a, c, e</td>
</tr>
<tr>
<td>5</td>
<td>c, d, e, f, k</td>
<td>c, d, e, f</td>
</tr>
<tr>
<td>6</td>
<td>b, c, d, e, f</td>
<td>b, c, d, e, f</td>
</tr>
<tr>
<td>7</td>
<td>b, d, e, f</td>
<td>b, d, e, f</td>
</tr>
<tr>
<td>8</td>
<td>e, f</td>
<td>e, f</td>
</tr>
</tbody>
</table>

Fig. 1. An example dataset.

<table>
<thead>
<tr>
<th>a b c d e f</th>
<th>rowSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>1 1 0 1 0 1</td>
<td>4</td>
</tr>
<tr>
<td>1 0 1 0 1 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 1 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td>0 1 1 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td>0 1 0 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 0 1 1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a b c d e f</th>
<th>rowSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>1 1 0 1 0 1</td>
<td>4</td>
</tr>
<tr>
<td>1 0 1 0 1 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 1 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td>0 1 1 1 1 1</td>
<td>5</td>
</tr>
<tr>
<td>0 1 0 1 1 1</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 0 1 1</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 2. Bit matrix of the dataset.

Fig. 3. Pruned bit matrix with minsup = 2.
In this paper we use the bit matrix representation of datasets to extract the complete set of frequent itemsets of datasets using an efficient parallel algorithm based on systolic arrays.

Here are the main contributions of this paper:

1. A new mining method is developed which uses divide and conquer approach to reduce size of bit matrix. Therefore it solves the mining problem for typical datasets efficiently.
2. A new approach to modeling the frequent pattern mining problem as a systolic array problem is presented.
3. A systolic array-based parallel method is designed to mine the itemsets of a dataset with using our bit wise approach to mine the frequent patterns from very large datasets efficiently.

The remaining of the paper is organized as follow:

In Section 2, we discuss the existing studies related to the parallelism of frequent pattern mining problem. We also describe existing serial pattern mining algorithms which are based on bit wise representation of dataset. Section 3 includes the defining and describing preliminaries of systolic arrays and represents a primary model of parallel frequent itemset mining as a systolic array problem. We present our new parallel method and its algorithm in Section 4 and conduct experimental study in Section 5. Finally we conclude the study in Section 6.

2. Related work

This section includes two parts: in the first sub-section we explain about recently-presented bit-wise mining algorithms and in the second one recent parallel and distributed pattern mining algorithms are described.

2.1. Bit-wise pattern mining methods

Recently, many attempts have been given to applying bitmap techniques in the frequent patterns mining algorithms [18–21]. BitTableFI [18] is such a recently proposed efficient BitTable-based algorithm, which exploits BitTable both horizontally and vertically. BitTable is a special data structure which is used horizontally and vertically to compress database for quick candidate itemsets generation and support count, respectively.

Although making use of efficient bit wise operations, BitTableFI still may suffer from the high cost of candidate generation and test. To address this problem, a new algorithm Index-BitTableFI was proposed [19]. Index-BitTableFI also uses BitTable horizontally and vertically. To make use of BitTable horizontally, index array and the corresponding computing method were proposed. By computing the subsume index, those itemsets that co-occur with representative item can be identified quickly by using breadth-first search at one time.

In [20] a fast mining algorithm was represented which integrates the merits of the matrix algorithm and Index-BitTableFI algorithm, and designs an efficient algorithm for mining the frequent itemsets. In this algorithm, it may generate directly some frequent itemsets which do not generate in the Index-BitTableFI. On the other hand, the algorithm does not use recursive method which is time-consuming to compute other frequent itemsets in Index-BitTableFI algorithm, and uses breadth-first search strategy to generate all frequent itemsets.

CFP-tree and CFP-array [21] are two novel data structures which use lightweight compression techniques to reduce memory consumption of FP-tree based algorithms by an order of magnitude. CFP-tree exploits a combination of structural changes to the FP-tree and bitmap techniques.

2.2. Parallel pattern mining methods

Several parallel and distributed algorithms have been proposed so far to mine all the frequent itemsets in a transaction database [22–27]. In 1999 a survey was conducted on association rule mining and its parallelization schemes [22]. In this survey, most of the first generation of parallel and distributed pattern mining algorithms, which were Apriori-based, were studied. All of these algorithms suffer hugely from the Apriori limitations since they are based on the Apriori principle.

Inspired by the performance gain of FP-growth over Apriori-based techniques, a large number of studies have exploited the FP-tree-based mining technique in parallel and distributed pattern mining [23–27]. Some of these works partition the database into several parts and then individually construct local FP-tree for each part in parallel [23,24]. On the other hand, there are some other algorithms which build a global FP-tree for the entire database, by parallelizing the tree-building technique [25,26].

An efficient parallel algorithm for mining frequent patterns on parallel shared nothing platforms has been presented in [23]. By efficiently partitioning the list of frequent items list over processors, this algorithm (PFP-tree) tries to introduce minimum communication and synchronization overheads. Load balancing FP-tree (LFP-tree) is another parallel and distributed mining algorithm based on FP-tree structure [24]. The algorithm divides the item set for mining by evaluating the tree’s width and depth. Moreover, a simple- and trusty-calculated formulation for loading degree was proposed in [24]. The communication time reduced in LFP-tree by...
preserving the heavy loading items in their local computing nodes and hence LFP-tree reduces the computation time and has less idle time compared with PFP-tree. In addition, it has better speed-up ratio than PFP-tree when number of processors grow.

Both the LFP-tree and the PFP-tree approaches broadcast the itemset-based prefix-trees to all processors multiple times. Instead of constructing several local FP-trees, Chen et al. [26] proposed a tree partition-based technique that builds only one FP-tree and partitions it into several independent parts to mine frequent patterns by assigning one part to one processor. However, this approach still suffers from memory constraint and inter-processor communications cost. DFP is a strategy for mining frequent item sets from terabyte-scale data sets on cluster systems which was presented in [25]. In this strategy some optimizations have been designed for lowering communication costs using compressed data structures and an encoding method. Optimizations for improving cache, memory and I/O utilization using pruning techniques, and smart data placement strategies are also employed in this strategy.

PP-tree (Parallel Pattern tree) is a novel tree structure which is proposed in [27]. PP-tree can reduce the I/O cost by capturing the database contents with a single scan and facilitates efficient FP-growth mining on it.

Even though the local or the global FP-trees can be constructed with less inter-processor communications, almost all approaches suffer from excessive communication overhead, during the mining process. All previously developed parallel pattern mining methods also have a common problem that is “how we can divide the data-set between computers such that they can do the mining process independently and without needing to share information and results”.

Fig. 7. Initial state of systolic array.

Fig. 8. Moving first input in the systolic array.
We use systolic array to solve the problem. Systolic arrays technique enables us to mine frequent pattern parallel without any need to share information and result and I/O overloading.

3. Problem modeling

In this section we introduce systolic arrays and model a parallel frequent itemset mining problem with them using the bit wise representation method.

3.1. Systolic arrays

A systolic array is an arrangement of processors in an array where data flows synchronously across the array between neighbors, usually with different data flowing in different directions.

Each processor at each step takes in data from one or more neighbors (e.g. North and West), processes it and, in the next step, outputs results in the opposite direction (South and East). Kung and Leiserson were the first to publish a paper on systolic arrays, and coined the name [29]. This is a specialized form of parallel computing in which multiple processors were connected by short wires. Cells (processors), compute data and store it independently from each other. In this type of parallelism, unlike many other forms of parallelism, we do not lose speed through the processor's connection. Fig. 4 shows an example of a systolic array cell. This cell has two inputs “a” and “b” and a memory “m” to store data. After each computation level, the cell provides two output values “x” and “y”. The computation formula has been determined for processor.

3.2. Modeling itemset mining problem using systolic arrays

To use systolic array as the implementation environment of our parallel algorithm we should first define values of inputs, outputs and the internal memory of cells. For this purpose, we first try to represent a systolic array which is able to construct all itemsets of a dataset and their corresponding bit vector to indicate whether

![Diagram of systolic array processing]
is it frequent or not. The primary idea is this: “determine frequent items in the dataset, if there exists “n” frequent item in the dataset, then there will be $2^n - 1$ potentially frequent itemset in it, so produce the binary form of each number from 1 to $2^n - 1$ with systolic array and then correspond each item to one bit of the number”. To implement this idea we consider a one dimensional systolic array with cells such as in Fig. 5.

Here are the specifications and rules of the cells of our systolic array:

- Each cell has a memory ($x$) which is storing an item’s bit vector.
- Each cell has a memory ($m$) which is storing an item’s name.
- Each cell has a memory ($z$) which is storing a bit. If there is the corresponding item of a cell in an itemset in a pass, then $z$ is 1 in that pass else $z$ is 0 (item’s presence situation bit).
- Each cell has an input ($y$) which is structured of three parts: $y$(IS) include an itemset, $y$(array) include the corresponding bit vector of the itemset, and $y$(sw) is a switch that determine if the pass is complete or not.
- Each cell has two outputs “$a$” and “$b$”.

In the first method of implementation, assume we have a one-dimensional array with $n$ cells ($n$ is the number of frequent items in the dataset) such that the output “$a$” of cell “$i” is the input “$y$” of cell “$i + 1$” (for each $0 < i < n$). Initial input for the first cell in each pass is a $y$ with an empty set as $y$(IS), a bit vector with $n$ bit which the value of each bit is set to 1 as $y$(array) and switch $y$(sw) with value 1. For example consider the 8-rows dataset of Fig. 2. If the minsup is 2 the 3rd column of Fig. 2 will be resulted. Therefore the set of frequent items is $\{a, b, c, d, e, f\}$ so the corresponding bit matrix is such Fig. 3. Since each cell of our systolic array should store the bit representation of an item, so we need the vertical representation of the dataset. The vertical dataset can be created by transposing the corresponding bit matrix. Fig. 6 shows the transposed representation of corresponding bit matrix of Fig. 3.

Each row of this transposed bit matrix is corresponding to an item’s bit vector and is storing in one of the cells of the systolic array. Fig. 7 shows the initial systolic array for this example.

For simplicity, in the systolic array of Fig. 7, we do not show the bit vector of the items of each cell. Fig. 7 only shows the item’s name and its presence situation bit of each cell and the corresponding bit vector of each cell can be considered in the transposed bit matrix of Fig. 6.

The mining idea is to construct the set of all possible itemsets and their corresponding bit vectors, and to test their frequencies by counting the number of the bits with value “1” in their corresponding bit vector. Since each column of the transposed bit matrix is corresponding to a row of the dataset, so if there are more than “minsup” bit(s) (or exactly “minsup” bit(s)) with value 1 in the corresponding bit vector of an itemset, then that itemset will be a frequent itemset.

The systolic array of Fig. 7 is going to construct binary representation of the numbers between 1 and 63 ($2^6 - 1$ for our six items) such that each bit of the constructed number stored in the one cell of array and then use the bits of constructed number to create a pattern. The created pattern and its corresponding bit vector moving between cells and a flag checks the end of pattern creating in each pass.

Computation process of each cell in each pass is based on these rules:
If \( y(sw) = 1 \) then
- If \( z = 1 \) then
  - Let \( y(array) = ((x) \text{ AND } (y(array))) \).
  - Let \( y(IS) = y(IS) \cup m \).
  - \( y(sw) \) remain 1.
  - Let \( a = y \).
  - Let \( b = \ast \).
- Else (if \( z = 0 \))
  - Let \( y(sw) = 0 \).
  - Let \( z = 1 \).
  - Let \( a = y \).
  - Let \( b = \ast \).

Else (if \( y(sw) = 0 \))
- If \( z = 1 \) then
  - Let \( y(array) = ((x) \text{ AND } (y(array))) \).
  - Let \( y(IS) = y(IS) \cup m \).
  - \( y(sw) \) remain 0.
  - Let \( a = y \).
  - Let \( b = \ast \).
- Else (if \( z = 0 \))
  - Let \( a = y \).
  - Let \( b = \ast \).

This computation process produces all number from 1 to \( 2^n - 1 \) in the cells of systolic array such that each bit of the number is stored in one cell of array. For example 6-bit binary representation of 31 is 011111 and this number store in the systolic array’s cells from left to right (0 in the left-most cell and 1s in the other cells).

You can see the trace of some pass in Figs. 8–10.

Fig. 8 shows that the itemset \( \{f\} \) is frequent because the number of 1 in the output \( y(array) \) is 7 and is greater than \( \text{minsup} \) (\( \text{minsup} = 2 \)).

Fig. 9 shows the pass of second input of systolic array over its cells.

You can see that after first input the bits of array’s cell constructed the binary form of 2 and after second input they constructed the binary form of 3. It is very important to note that know after first input passing the first cell the second input is ready and enters instantaneously and after entering first input to third cell and second one to second cell, third input enters in first cell and so on.

Fig. 10 shows some movements of input \( y \) in the systolic array in the third pass.

These figures show the function of this systolic array.

We can see in figures that after passing from left-most “1” of the array no change occurs on cell’s output. Since only \( \log k + 1 \) (for each \( k \)) right bit of binary form of \( k \) is used so we can use a counter and stop each movement after passing the input from left-most “1” of the systolic array.

4. Parallel mining method

In this section we present a parallel mining method. First we describe how we can mine the dataset using divide and conquer method serially and then we present our parallel algorithm.
We can use this approach in parallel frequent pattern mining to improve execution time. For this aim we assign each above divided
group above to a systolic array and use the array to construct the
patterns of each group. So we can increase the level of parallelism
in our method.

4.2. Two-dimensional systolic array

Based on divide and conquer approach, we can use a systolic ar-
ray with \( n - 1 \) cells to mine first category of itemsets (frequent
itemsets which contain \( a \)). Fig. 11 shows the structure of this array.

In this array we use the same structure of arrays in Section 4 but
the input includes item “\( a \)” and array has not any cell correspond-
ing to item “\( a \)”. The input value will be moved in the array and cells
will do their operation just like array in Section 4. We can develop
\( n - 1 \) similar arrays to \( n - 1 \) Items (all items but the last) of data-
set. So we have a two-dimensional array like Fig. 12 which reduces
the execution time very much.

Since we continue the operations of each pass of each row of the
array while the input arrive to the left-most “1” of the systolic ar-
ray, the execution time is improved in many cases. Furthermore we
know if the number of “1” in \( y \) (array) is less than minsup the
process of that \( y \) will be stopped. So we can sort the items by their
frequency and increase the probability of stopping process in the
higher rows of array which have more cells and so gain a new
improvement of our algorithm.

Our experimental results show that these improvements
incredibly reduce the executing time of mining algorithm.

4.3. Two-dimensional systolic array

Through using the divide and conquer approach, we represent a
novel parallel algorithm, SABMA (Systolic Array Based Mining
Algorithm), which mines the frequent patterns of a dataset effi-
ciently. SABMA uses a two-dimensional systolic array which has
\( n - 1 \) rows, with each row “\( k \)” \((\forall 1 \leq k \leq n - 1)\) containing “\( n-k \)”
cells. Fig. 13 shows the Systolic Array Based Mining Algorithm
(SABMA).

Input parameters of this algorithm include \( m \), number of rows
of dataset (also the number of the columns of transposed bit ma-
trix), \( n \), number of frequent items of the dataset (also the number
of the rows of transposed bit matrix), Items, list of all frequent
items, \( y \), input of each of the systolic array’s rows, minsup, the
(user-defined) minimum support of patterns and bitMat, trans-
posed bit matrix of our dataset.

In the main part (in the main ‘while’ block of algorithm), the
algorithm constructs the inputs for different rows of systolic array,
repeatedly, and sends them to the first cell. In each round of the
loop, the algorithm also checks the output of the last cell to output
the constructed value to file.

In each round of internal loop (for), first the input \( y \) is initiated
by corresponding item value. For example when we have six items

```plaintext
InputStruct = Record
Begin
Sw: boolean;
Is: string;
bitVector: array [1..m] of boolean;
End;
Procedure SABMA
Input
m: integer; // number of rows of dataset
n: integer; // number of frequent items
items: array [1..n] of integer; // set of all frequent items
y: array [1..n-1] of InputStruct; // input of systolic array's rows
minsup: integer; // minimum support
bitMat: array[1..n][1..m] of Boolean;
Output
File: TextFile; // File of the all frequent itemsets mined from the dataset
Var
i: integer;
Begin
t=1;
while (there is at list one bit z with value 0) then
Begin
For i := 1 to n-1 do
Begin
y[i].Is := intosr(items[i] + ’’);
y[i].bitVector := bitmat[items[i]];
y[i].sw := 1;
send y[i] to first cell of row i of systolic array;
if (there are a valid value in output ‘a’ of cell \( \left[ \log_2 \right] +1 \)) then
if (the itemset is frequent) then output (File, a.Is)
t=t+1;
End;
End;
End;
```

Fig. 13. SABMA algorithm.
(a, b, c, d, e, f) in our dataset, n is 6 and the internal loop has 5 iteration for 5 rows. In the first iteration, first $y[1]$ is initiated by values $y[1]_{Is='a'}$, $y[1]_{bitVector} = 10110000$ (the corresponding row of item ‘a’ in the bit matrix), and $y[1]_{sw} = 1$ for first row. Similarly, next iterations prepare the corresponding inputs for next rows. After input initialization, the constructed $y[1]$ move from one cell to another cell across the first row of systolic array. When this input arrives to the cell number $\log_2(2 + 1) = 2$, its corresponding itemset’s (ce) frequency will be checked by the algorithm, and ‘ce’ will be written in the output file as a mined frequent itemset. Based on systolic array rules, after this pass, in the third row of the systolic array, the $z$ bit of cells d, e, and f will be 0, 1, and 1 respectively. These z-values prepare this row of systolic array for the next input which will construct the next itemset that contain item c (cef). The external loop (while) ensures constructing all itemset in each row of systolic array.

5. Experimental results

In this section we will study the performance of our algorithm. Since our algorithm extract the set of all frequent itemsets of dataset, we have maximum accuracy (100%) in extracted patterns. Therefore we do not present performance comparison regarding accuracy and size of the output patterns and we compare the parallel methods performance in executing time. All our experiments were performed on a computer with a 2.2 GHz core2 duo CPU, 2 GB RAM and 120 GB hard disk. All the reported runtime include both computation time and IO time. Although PP-tree [27] has already

<table>
<thead>
<tr>
<th>datasets</th>
<th>size</th>
<th>Transactions#</th>
<th>Items#</th>
</tr>
</thead>
<tbody>
<tr>
<td>mushroom</td>
<td>0.56M</td>
<td>8124</td>
<td>119</td>
</tr>
<tr>
<td>pumsb</td>
<td>16.3M</td>
<td>49046</td>
<td>2113</td>
</tr>
<tr>
<td>Accident</td>
<td>33.85M</td>
<td>340183</td>
<td>468</td>
</tr>
<tr>
<td>Retail</td>
<td>3.97M</td>
<td>88162</td>
<td>16469</td>
</tr>
<tr>
<td>kosarak</td>
<td>30.5M</td>
<td>990022</td>
<td>41270</td>
</tr>
</tbody>
</table>

Fig. 14. Characteristics of test datasets.

![Fig. 15. Dataset mushroom.](image1)

![Fig. 16. Dataset pumsb.](image2)

![Fig. 17. Dataset accident.](image3)

![Fig. 18. Dataset retail.](image4)

![Fig. 19. Dataset kosarak.](image5)
shown its better performance than other column enumeration and row enumeration based parallel algorithms such as PFP-tree [23], LFP-tree [24], and partition-based parallel techniques [25,26], it was considering other experimental configuration in previous works. Therefore, we compare our algorithms with PP-tree based parallel mining algorithm (as the best reported parallel algorithm), LFP-tree based parallel mining algorithm (as the best reported shared nothing platform parallel algorithm), and DFP (as the best reported partition based parallel technique).

Because of the variant size of the datasets used for performance study, we do not determine the minsup threshold with an absolute number. Instead, minsup is determined by a percentage of the number of transactions. We use five real datasets to compare the algorithms. We use five real standard datasets from FIMI [28] to compare the algorithms. Fig. 14 shows some statistical information about the dataset used for performance study.

Note that the number of processors which is simulated by our software for each of two algorithms is equal in each running.

Figs. 15–19 show the result of running two algorithms SABMA (our systolic array based parallel algorithm) and PP-tree (parallel pattern tree based algorithm) on real standard datasets. We can see that with increasing minimum support all the algorithm performance in all dataset will be decreased.

6. Conclusion

One of the major problems in frequent itemset mining is the explosion of the number of results which is directly affecting on the execution time of itemset mining algorithms, specially, for high dimensional and large datasets. A useful and suitable way to address this problem is using parallel and distributed techniques. In this paper we represent a parallel method to conduct mining of frequent patterns in very large datasets. In our method, we use a bit matrix to compress the dataset and make it easy to use in mining algorithm and prepare the infrastructure for parallelism. We use the systolic arrays to parallel frequent pattern mining because of its characteristic such as very low cost of connection and computation. Our experimental result shows that our algorithm attains very good mining efficiencies on various input datasets. Furthermore, our performance study shows that this algorithm outperforms substantially the best previously developed algorithms.

References

[5] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceeding of the 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, TX, 2000, pp. 1–12.