

# Parallelizing Frequent Itemset Mining with FP-Trees

Peiyi Tang

Department of Computer Science  
University of Arkansas at Little Rock  
Little Rock, AR 72204

Markus P. Turkia

Department of Computer Science  
University of Arkansas at Little Rock  
Little Rock, AR 72204

## Abstract

A new scheme to parallelize frequent itemset mining algorithms is proposed. By using the extended conditional databases and  $k$ -prefix search space partitioning, our new scheme can create more parallel tasks with better balanced execution times. An implementation of the new scheme with FP-trees is presented. The results of the experimental evaluation showing the increased speedup are presented.

## 1 Introduction

The fundamental problem in mining association rules from a large database is to discover all the frequent itemsets. A frequent itemset is a set of items whose support is greater than or equal to a given threshold. The support of an itemset is the number of transactions in the database containing it. The search space of frequent itemsets is the set of all possible itemsets, the power set of the total set of all (frequent) items in the database. Therefore, the search space grows exponentially as the size of the total set of items increases. Another factor affecting the time complexity of frequent itemset mining is the size of the database: the number of transactions in the database. Especially when the threshold is low, the number of transactions to be considered is large for large databases in the real world. As indicated by the performance measurement posted in the FIMI repository [1], the frequent itemset mining times of all algorithms for all datasets grow exponentially as the threshold decreases.

Since the problem of data mining of association rules was formulated in 1993 [2], numerous sequential algorithms and implementations running on a single processor were proposed. Due to the exponential complexity of the problem, it is desirable to use parallel processors to speedup the time-consuming computation. Parallel data mining has been studied extensively in [3, 4, 5, 6]. All these are trying to parallelize the *Apriori* algorithm [2]. In this paper, we propose a parallelization scheme which can be used to

parallelize the efficient and fast frequent itemset mining algorithms based on frequent pattern trees (FP-trees) [7]. To achieve high speedup of parallel mining of frequent itemsets, it is important to reduce the execution time of the largest parallel task. In this paper, we extend the concepts of conditional databases of [7] and  $k$ -prefix search space partitioning [8] to allow more parallel tasks. Our scheme is able to reduce the execution time of the largest parallel task, thus allowing more parallel processors to join to achieve higher speedup. We also use dynamic self-scheduling [9] for parallel task allocation to balance the work load among parallel processors. The experimental results show that our parallelization scheme can achieve the high speedup.

The organization of the rest of the paper is as follows. Section 2 provides the preliminaries of basic concepts and their notations to facilitate the discussion. Section 3 describes  $k$ -prefix partitioning of search space and conditional databases. Section 4 describes the formation and dynamic allocation of parallel tasks and our parallel mining algorithm. Section 5 provides the results of experimental evaluation. Section 6 concludes the paper.

## 2 Preliminaries and Notations

A database  $D$  is a bag<sup>1</sup> of transactions. A transaction is a set of items which is a subset of the *total set of items*  $I = \{i_1, \dots, i_n\}$ , the set of all items used in the database. An *itemset* is also a subset of the total set of items  $I$ . The *support* of itemset  $x$  in database  $D$ , denoted as  $Sup_D(x)$ , is the number of transactions in  $D$  containing  $x$ . That is,

$$Sup_D(x) = |\{t \mid t \in D \wedge x \subseteq t\}| \quad (1)$$

Given a threshold  $1 \leq \xi \leq |D|$ , an itemset  $x$  is *frequent* if  $Sup_D(x) \geq \xi$ . The frequent itemset mining problem is to find all the itemsets  $x$  with  $Sup_D(x) \geq \xi$  for a

---

<sup>1</sup>A bag is a set which allows multiple occurrences of an element.

given threshold  $\xi$ . A subset of any frequent itemset is also frequent, because  $Sup_D(y) \geq Sup_D(x)$  if  $y \subseteq x$ , according to (1). An item  $i$  is frequent, if the support of its singleton set is greater or equal to the threshold, i.e.  $Sup_D(\{i\}) \geq \xi$ . Obviously, non-empty frequent itemsets can only be made of *frequent items*.

The support of the empty itemset  $\emptyset$  is  $|D|$ , i.e.  $Sup_D(\emptyset) = |D|$ , because every transaction in  $D$  contains  $\emptyset$ . Thus, the empty itemset  $\emptyset$  is a trivial frequent itemset due to  $|D| \geq \xi$ .

Without loss of generality, we can assume that  $I$  is the set of frequent items of database  $D$  [10]. We also assume that the items of  $I$  are sorted according to the increasing order of support, i.e.  $Sup_D(\{i_1\}) \leq \dots \leq Sup_D(\{i_n\})$ .

In this paper, we use strings to represent itemsets. Therefore, the total set of items is written as  $I = i_1i_2 \dots i_n$ . The search space is the power set of  $I$  denoted as  $2^I$ . It is a complete Boolean lattice,  $\langle 2^I, \cup, \cap \rangle$ . Any subset of  $I$  will be written as a string of items in the increasing order of their supports. The contiguous substrings of  $I$  will be written as  $I[i..j]$  with  $1 \leq i < j \leq n$ . In particular, the  $k$ -prefix of  $I$  is denoted as  $I_k = I[1..k]$ . The empty subset of  $I$  is denoted as  $\epsilon$ .

### 3 Mining Conditional Databases

To parallelize frequent itemset mining, we need to partition the search space first. The depth-first search algorithm proposed in [7] can find all the frequent itemsets (FI) without enumerating all the candidates in  $2^I$ . This is the reason why the mining algorithm of [7] is faster than the old breadth-first algorithms based on the *Apriori* approach [2].

Given  $I = abcde$ , the algorithm of [7] using FP-trees is to find all the frequent itemsets containing  $a$  first, then those containing  $b$  but not  $a$ , those containing  $c$  but not  $ab$ , and so on.

Obviously, the number of partitions is  $n$ , the number of frequent items. For parallel processing using large number of processors, the load balancing is important. Larger number of partitions for larger number of tasks tend to balance the work load among parallel processors better. As proposed in [8], we can partition the lattice based on the power set of the  $k$ -prefix  $I_k$ . In particular, for each itemset  $x$  from the power set of  $I_k$  (i.e.  $x \in 2^{I_k}$ ), we define the partition  $P_x$  as follows:

$$P_x = \{x \cup y \mid y \in 2^{I[(k+1)..n]}\} \quad (2)$$

Obviously,  $P_\epsilon = 2^{I[(k+1)..n]}$  and

$$P_{x_1} \cap P_{x_2} = \emptyset, \forall x_1, x_2 \in 2^{I[1:k]} \wedge (x_1 \neq x_2)$$

By using  $k$  items from  $I$ , we are able to have  $2^k$  partitions. Partition  $P_\epsilon$  will continue to be partitioned which allows to have totally as large as  $\lfloor \frac{n}{k} \rfloor (2^k - 1) + (2^{n \bmod k} - 1)$  parallel tasks as we will see later.

To mine the frequent itemsets in a particular partition  $P_x$ , we do not need the whole database, but only the transactions of  $D$  containing  $x$ . We now extend the conditional database of [7] and redefine it as follows:

**Definition 1** Given a database  $D$  and a subset of prefix  $I_k$ ,  $x \in 2^{I_k}$ , the conditional database of  $D$  on  $x$ , denoted as  $D_x$ , is

$$D_x = \{t - I_k \mid x \subseteq t \wedge t \in D\}$$

Here, the set subtraction  $t - I_k$  is  $t \cap \overline{I_k} = t \cap I[(k+1)..n]$ .

It is important to note that conditional database  $D_x$  may have empty transactions  $\epsilon$ . This happens when the original database  $D$  has transactions that contain  $x$  and contain the items in  $I_k$  only. It is important to include these empty transactions in  $D_x$ , because they will affect the decision on if  $x$  is a frequent itemset of  $D$  as we will see later.

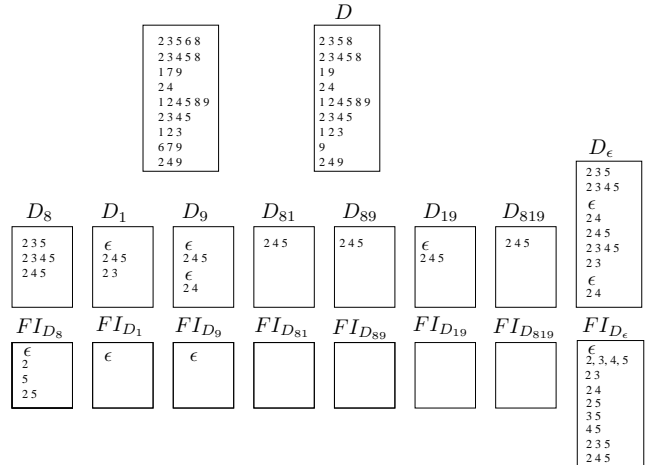


Figure 1: Conditional Databases and Their FIs ( $\xi = 3$ )

Figure 1 shows an example of database and its conditional databases. The original database at the top-left shows that it has 9 transactions using 9 items named  $1, \dots, 9$ . Suppose that the threshold is  $\xi = 3$  and the frequent items in the non-decreasing order of support are:  $8(3)$ ,  $1(3)$ ,  $9(4)$ ,  $5(4)$ ,  $3(4)$ ,  $4(5)$ ,  $2(7)$ . The numbers in the parentheses above are the supports of the items. The database containing only these frequent items is shown as  $D$ . Therefore, the total set of (frequent) items is  $I = 8195342$ . Suppose that  $k = 3$ . So  $I_k = 819$  and we have eight conditional databases:  $D_8, D_1, D_9, D_{81}, D_{89}, D_{19}, D_{819}$  and  $D_\epsilon$ , as shown in Figure 1. Note the empty transactions  $\epsilon$  in  $D_1, D_9, D_{19}$

and  $D_\epsilon$ . For example,  $D$  has three transactions containing 1: 19, 124589 and 123. Subtracting 819 from each of them gives the three transactions in  $D_1$ :  $\epsilon$ , 2345 and 23, respectively.

According to Definition 1, we can prove [10] that the support of any itemset  $y \subseteq I - I_k$  in  $D_x$  is the same as the support of itemset  $x \cup y$  in  $D$ :

$$Sup_{D_x}(y) = Sup_D(x \cup y)$$

Therefore, for an itemset  $y \subseteq I - I_k$ , itemset  $x \cup y$  is frequent in  $D$  if and only if itemset  $y$  is frequent in  $D_x$ .

Based on the discussion above, the mining of frequent itemsets for  $D$  can be reduced to mining the frequent itemsets for each conditional database  $D_x$  for all  $x \in 2^{I_k}$ . This is summarized in the following theorem.

Let the sets of frequent itemsets of  $D$  and  $D_x$  be denoted as  $FI_D$  and  $FI_{D_x}$ , respectively.

**Theorem 1** *Given a database  $D$  with the total set of items  $I$ , a threshold  $\xi$ , and the  $k$ -prefix  $I_k$  of  $I$ , the set of frequent itemsets of  $D$ , can be partitioned to the sets of frequent itemsets mined from each individual conditional database  $D_x$  for all  $x \in 2^{I_k}$  as follows:*

$$FI_D = \bigcup_{x \in 2^{I_k}} FI'_{D_x}$$

and

$$FI'_{D_{x_1}} \cap FI'_{D_{x_2}} = \emptyset$$

for  $x_1 \neq x_2$ , where  $FI'_{D_x}$  is the set of frequent itemsets mined from  $D_x$  as  $FI'_{D_x} = \{x \cup y \mid y \in FI_{D_x}\}$ .

When applying Theorem 1 to mine frequent itemsets of  $D$ , it is important to consider if the empty itemset  $\epsilon$  is frequent or not in each conditional database  $D_x$ , because  $x$  is a frequent itemset of  $D$  if and only if  $\epsilon$  is frequent in  $D_x$ . The support of the empty itemset  $\epsilon$  is the total number of transactions in the database, because every transaction contains it. Therefore, empty itemset  $\epsilon$  is frequent in  $D_x$  if and only if there are at least  $\xi$  transactions (including empty transactions  $\epsilon$ ) in  $D_x$ . In the conditional databases shown in Figure 1, empty itemset  $\epsilon$  is frequent only for  $D_8, D_1, D_9$  and  $D_\epsilon$ , because each of them has at least  $\xi = 3$  transactions. The set of frequent itemsets of each conditional database in the example is also shown in Figure 1. Note that  $FI_{D_{81}}, FI_{D_{89}}, FI_{D_{19}}$  and  $FI_{D_{819}}$  are empty, but  $FI_{D_1}$  and  $FI_{D_9}$  are not empty, as they contain  $\epsilon$ . As a result,  $FI'_{D_{81}}, FI'_{D_{89}}, FI'_{D_{19}}$  and  $FI'_{D_{819}}$  are empty, but  $FI'_{D_1} = \{1\}$  and  $FI'_{D_9} = \{9\}$ . We also have  $FI'_{D_8} = \{8, 82, 85, 825\}$  because  $FI_{D_8} = \{\epsilon, 2, 5, 25\}$ , and  $FI'_{D_\epsilon} = FI_{D_\epsilon} = \{\epsilon, 2, 3, 4, 5, 23, 24, 25, 35, 45, 235, 245\}$ .

Finally, the set of frequent itemsets of  $D$  is the union of these sets of frequent itemsets mined from the conditional databases:  $FI_D = \{8, 82, 85, 825, 1, 9, \epsilon, 2, 3, 4, 5, 23, 24, 25, 35, 45, 235, 245\}$

## 4 Parallel Tasks and Their Allocation

To have more parallel tasks, we need to continue to partition the job of mining  $D_\epsilon$  using the same mechanism as the original database  $D$ . Now the problem becomes to mine the database  $D_\epsilon$  with the total set of items  $I - I_k$  using its  $k$ -prefix  $I[(k+1)..2k]$ . Again, according to Theorem 1, the mining of  $D_\epsilon$  can be partitioned to mining each individual  $D_{\epsilon_x}$  for all non-empty  $x \in 2^{I[(k+1)..2k]}$  in parallel on multiple processors. This process will continue until the total set of items becomes empty. In general, given the original set of frequent items  $I = i_1 i_2 \dots i_n$  and integer  $1 \leq k \leq n$ , the data mining of frequent itemsets of  $D$  is divided to  $\lceil \frac{n}{k} \rceil$  rounds of parallel tasks. Each round consists of  $2^k - 1$  tasks with possible exception for the last round. The number of tasks of the last round is  $2^{n \bmod k} - 1$ , if  $k$  does not divide  $n$ . Therefore, the total number of parallel tasks is

$$P(n, k) = \lfloor \frac{n}{k} \rfloor (2^k - 1) + (2^{n \bmod k} - 1) \quad (3)$$

Each parallel task is denoted as  $T_{j,l}$  with  $1 \leq j \leq \lceil \frac{n}{k} \rceil$  and  $1 \leq l \leq 2^k - 1$ . For the database  $D$  with  $I = 8195342$  shown in Figure 1, there are 15 parallel tasks with  $k = 3$ , namely,  $T_{1,1}, \dots, T_{1,7}, T_{2,1}, \dots, T_{2,7}$  and  $T_{3,1}$ . Let the binary code of integer  $l$  be denoted as  $b(l)$ . The job of task  $T_{j,l}$  is to mine the frequent itemsets from conditional database  $D_{j,x}$  where  $x = I[(j-1)k+1..jk] \hat{\wedge} b(l)$ . Here, operator  $\hat{\wedge}$  between a  $k$ -string (ordered set)  $S = s_1 \dots s_k$  and a binary  $k$ -vector  $B = b_1 \dots b_k$  is defined as  $T = S \hat{\wedge} B$  such that

$$\begin{cases} s_i \in T & \text{if } b_i = 1 \\ s_i \notin T & \text{if } b_i = 0 \end{cases}$$

for  $1 \leq i \leq k$ . The conditional database  $D_{j,x}$  is

$$D_{j,x} = \{t - I[(j-1)k+1..jk] \mid x \subseteq t \wedge t \in D_{j-1}\} \quad (4)$$

where  $D_{j-1}$  is the  $\epsilon$  conditional database to be partitioned for parallel processing and it is as follows according to the discussion above:

$$D_{j-1} = \{t - I[1..(j-1)k] \mid \epsilon \subseteq t \wedge t \in D\} \quad (5)$$

Note that  $D_0 = D$  and the  $D_\epsilon$  mentioned in section 3 is  $D_1$ .

The algorithm for task  $T_{j,l}$  is shown in function  $T(j, l)$  in Figure 2. Firstly,  $x$  is extracted from  $I$  according to the values of  $j$  and  $l$ . Then, the conditional

```

function  $T(j, l)$ 
begin
  if  $(j \leq \lfloor \frac{n}{k} \rfloor)$  then
    Let  $B$  be the  $k$ -bit binary vector of  $l$ ;
     $x \leftarrow I[\lfloor ((j-1)k+1)..jk \rfloor \overset{\circ}{\wedge} B]$ ;
  else
    Let  $B$  be the  $(n \bmod k)$ -bit binary vector of  $l$ ;
     $x \leftarrow I[\lfloor ((j-1)k+1)..n \rfloor \overset{\circ}{\wedge} B]$ ;
  endif
  Construct conditional database  $D_{j,x}$  from  $D$ 
  using (4) and (5);
  Call a frequent itemset mining algorithm to obtain
   $FI_{D_{j,x}}$  from  $D_{j,x}$  using  $\xi$ ;
  return  $\{x \cup y \mid y \in FI_{D_{j,x}}\}$ ;
end

main( )
begin
  if  $(p = 0)$  then
    Read threshold  $\xi$  and parameter  $k$ ;
    Broadcast  $\xi$  and  $k$  to all processors with  $p \neq 0$ ;
     $tasks \leftarrow \lfloor \frac{n}{k} \rfloor (2^k - 1) + 2^{n \bmod k} - 1$ ;
     $ticket \leftarrow 1$ ;
     $completed \leftarrow 0$ ;
     $totalFI \leftarrow \{\epsilon\}$ ;
    while  $(ticket \leq tasks \text{ or } completed < tasks)$  do
      receive a message from any processor  $Q \neq 0$ ;
      if (the message is a set of frequent itemsets  $FI$ )
        then
           $totalFI \leftarrow totalFI \cup FI$ ;
           $completed \leftarrow completed + 1$ ;
        endif
      send  $ticket$  to processor  $Q$ ;
       $ticket \leftarrow ticket + 1$ ;
    endwhile
    output  $totalFI$  to a file;
  else
    input database to construct  $D$ ,  $I$  and  $n$ ;
    receive  $\xi$  and  $k$  from processor 0;
     $tasks \leftarrow \lfloor \frac{n}{k} \rfloor (2^k - 1) + 2^{n \bmod k} - 1$ ;
    send an initial request to processor 0;
    receive  $ticket$  from processor 0;
    while  $(ticket \leq tasks)$  do
       $j \leftarrow \lceil \frac{ticket}{2^k - 1} \rceil$ ;
       $l \leftarrow ((ticket - 1) \bmod (2^k - 1)) + 1$ ;
       $FI \leftarrow T(j, l)$ ;
      send  $FI$  to processor 0;
      receive  $ticket$  from processor 0;
    endwhile
  endif
end

```

Figure 2: Parallel Algorithm of FI Mining

database  $D_{j,x}$  is constructed according to (4) and (5). Then, a frequent itemset mining algorithm is called to obtain the set of frequent items,  $FI_{D_{j,x}}$ , from  $D_{j,x}$  using threshold  $\xi$ . Lastly, each frequent itemset in  $FI_{D_{j,x}}$  is concatenated with  $x$  to form a frequent itemset of  $D$  to be sent to the master processor.

Since the execution times of parallel tasks  $T_{j,l}$  vary depending on the data in the database, it is best to use dynamic self-scheduling [9] to allocate tasks  $T_{j,l}$  to parallel processors at run-time. We dedicate processor 0 as the master processor for I/O and self-scheduling. The rest processors are work processors for parallel tasks. We use a single variable called *ticket* and initialize it to 1 in processor 0. This variable is incremented every time its value is sent to a work processor. The work processor will convert the received ticket to the index values of  $(j, l)$  for the task. Each work processor keeps requesting the next ticket after completing a parallel task until it receives a ticket larger than the total number of tasks  $P(n, k)$  (see (3)).

The SPMD (Single Program Multiple Data) parallel program for all processors is also shown in Figure 2.  $p$  is the processor number of each processor. The **then** part is executed by processor 0 and the **else** part by all other work processors.

We now discuss how to construct conditional database  $D_{j,x}$  required in function  $T(j, l)$  using FP-tree representation. Figure 3 shows the FP-tree of the example database  $D$  in Figure 1 according to [7]. At the left is the header table of all frequent items with non-decreasing support from bottom up. In each entry of the header table, the first number is the name of the item and the second number after colon its support. At the right of the figure is the FP-tree of the database. At each node, the first number is the name of the item and the second number after colon the number of occurrences of the item. The dash arrows show the links of the nodes representing the same item.

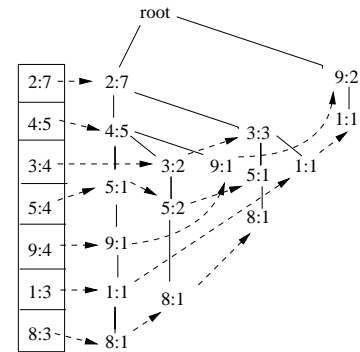


Figure 3: FP-Tree of Example Database

Given database  $D$ , the construction of  $D_{j-1}$  defined by (5) is straightforward:  $D_{j-1}$  is a subtree of

$D$  linked from the upper part of the header table by ignoring the first  $(j - 1)k$  items from the bottom. In our example in Figure 3,  $D_1$  is the subtree of  $D$  linked from items 5, 3, 4 and 2 only. This is the upper part of the tree obtained by trimming out the nodes linked from items 8, 1 and 9.

The construction of  $D_{j,x}$  defined by (4) can be obtained by extending the algorithm in [7] as follows. Assume  $x = x_1 \cdots x_p \in 2^{I[(j-1)k+1..jk]}$  with  $p \leq k$ . From each node linked from  $x_1$ , we traverse its ancestors following the parent node link. The traversal continues only if none of the  $x_2, \dots, x_p$  is missing. If the traversal has visited the nodes of all  $x_2, \dots, x_p$ , we have found a frequent pattern for  $D_{j,x}$ . This pattern contains all the ancestor items not in  $I[(j - 1)k + 1..jk]$  and the number of occurrences of each of them is that of  $x_1$ . For example, the conditional database  $D_{81}$  in Figure 1 corresponds to  $D_{1,81}$ . To construct  $D_{1,81}$  from the FP-tree in Figure 3, the algorithm follows the link from item 8 of the header table and finds three leaves for item 8. The traversal of the ancestors of the first leaf continues until it finds item 1. Therefore, it continues the traversal and finds the frequent pattern (5:1, 4:1, 2:1) (using the notation in [7]). The traversals from the second and third leaves of item 8 stop when they see nodes for item 5 without having visited any node for item 1. They do not provide any frequent pattern for  $D_{1,81}$ . Thus,  $D_{1,81}$  contains one itemset 245.

## 5 Experimental Evaluation

We have implemented the parallel mining algorithm in Figure 2 using FP-tree as the database representation. The code for sequential frequent itemset mining needed in task  $T_{j,l}$  is taken from [11], which is an implementation of the FP-growth algorithm of [7].

The parallel program runs on a cluster of PC workstations using MPI. Each processor is a Dell OptiPlex GX1p with Intel Pentium III (Katmai) processor of 497 MHz CPU, 512KB cache and 256MB RAM. The test runs for the datasets from FIMI repository [1] confirm that our parallel data mining algorithm and implementation is correct: the frequent itemsets obtained by the parallel processors are exactly the same by the sequential code [11].

To evaluate the performance of parallel processing and the impact of  $k$  value on the speedup, we run our parallel program on two data sets: T40I10D100K called T40 and **chess** from [1] with different  $k$  values.

T40 is a large dataset with about 1000 items and 100,000 transactions. We run the parallel code for T40 with threshold 500 on 1 to 16 work processors. The number of frequent items is 838. We have run the code

with four different values of  $k = 1, 2, 3, 4$ .

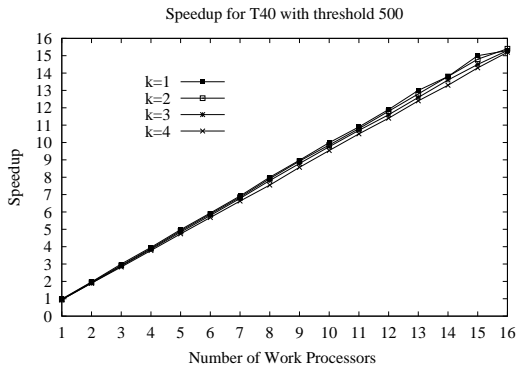
We also run all the parallel tasks on processor 0 to get the sequential execution times for  $k = 1, 2, 3, 4$ .

We use the sequential time of  $k = 1$  as the reference time, denoted as  $E_1$ , to calculate the speedup of parallel execution with  $P$  work processors as follows:

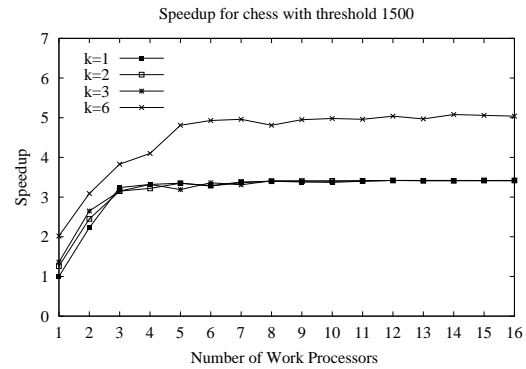
$$SP_P = \frac{E_1}{E_P}$$

Here,  $E_P$  is the parallel execution time measured on processor 0 after all parallel tasks are completed by  $P$  work processors. The speedups of T40 are shown in Figure 4(a). The frequent itemsets mined by work processors are not sent to processor 0. The time for building the FP-tree for the original database  $D$  is not included in  $E_1$  or  $E_P$ . The speedups of T40 plotted in Figure 4(a) show that the linear speedup is achieved and there are not much difference among the speedups of different  $k$  values. For  $k = 1$ , there are 838 tasks. With maximum 16 processors, the work load seems to be well balanced. We also measured the execution time of each parallel task. When  $k = 1$ , task  $T_{114,1}$  (ticket 114) has the highest execution time of 132.948 seconds. The sequential time is  $E_1 = 2060.155$  seconds. Therefore, the maximum speedup achievable is 15.49 ( $2060.155/132.948$ ). This means that we just reach this limit with 16 processors and more processors beyond 16 will not give us more speedup if  $k$  remains 1. In order to achieve higher speedup, we need to increase the  $k$  value to have more and smaller tasks and break that largest task, as we can see in the database **chess** below.

**chess** has 75 items and 3,196 transactions. We set threshold  $\xi$  to 1500 and there are 38 frequent items. We use the sequential time of  $k = 1$  as the reference sequential time again, which is 4904.113 seconds. Figure 4(b) shows the speedups of the parallel executions of **chess** on 1 to 16 work parallel processors. The results show that there are no significant differences among the performances for  $k = 1, 2, 3$ . When  $k$  reaches 6, the speedup shows significant improvement. This improvement is brought by two factors: (1) the sequential time is reduced from 4904.113 seconds ( $k = 1$ ) to 2420.633 seconds ( $k = 6$ ), and (2) the workload is better balanced with more tasks. When  $k = 1$ , there are 38 tasks and the largest task is task  $T_{15,1}$  (ticket number 15). It takes 1554.231 seconds to complete. Therefore, the maximum speedup achievable is 3.16 ( $4904.113/1554.231$ ). This is the reason for the low speedup for  $k = 1$  shown in Figure 4(b). This large task was not broken up until we increased the value of  $k$  to 6. There are 381 tasks when  $k = 6$  and the execution time of the largest task (task  $T_{3,32}$ , ticket number 158) is 967.349 seconds, lifting the maximum speedup to 5.07 ( $4904.113/967.349$ ). This max-



(a) Speedups of T40



(b) Speedups of chess

Figure 4: Speedups of Parallel Execution

imum speedup is reached approximately when 5 work processors are used (see Figure 4(b)).

## 6 Conclusion

We have presented a scheme to parallelize frequent itemset mining algorithms. By using the conditional databases extended from [7] and  $k$ -prefix partitioning from [8], our parallelization scheme is able to create more parallel tasks with better balanced execution times. We have implemented our scheme using FP-tree as the database representation and conducted preliminary experiments. The experimental results show that large  $k$  values will break large tasks to allow higher speedup.

## References

- [1] Bart Goethals and Mohammed J. Zaki. FIMI'03: Workshop on frequent itemset mining implementations. Technical report, <http://fimi.cs.helsinki.fi/>, 2003.
- [2] R. Agrawal, T. Imilinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Database*, pages 207–216, 1993.
- [3] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Transactions On Knowledge And Data Engineering*, 8:962–969, 1996.
- [4] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multiprocessors. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*. Article No. 43, 1996.
- [5] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.
- [6] Dejiang Jin and Sotirios G. Ziavras. A super-programming approach for mining association rules in parallel on pc clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15:783–794, 2004.
- [7] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International on Management of Data*, pages 1–12. ACM Press, 2000.
- [8] Peiyi Tang, Li Ning, and Ningning Wu. Domain and data partitioning for parallel mining of frequent closed itemsets. In *Proceedings of the 43<sup>rd</sup> Annual Association for Computing Machinery Southeast Conference (ACMSE'05)*, volume 1, pages 250–255, Atlanta, USA, March 2005.
- [9] Peiyi Tang and Pen-Chung Yew. Processor self-scheduling for multiple-nested parallel loops. In *Proceedings of the 1986 International Conference on Parallel Processing (ICPP'86)*, pages 528–535, August 1986.
- [10] Peiyi Tang and Markus P. Turkia. Parallelizing frequent itemset mining with FP-trees. Technical Report [titus.compsci.ualr.edu/~ptang/papers/par-fi.pdf](http://titus.compsci.ualr.edu/~ptang/papers/par-fi.pdf), Department of Computer Science, University of Arkansas at Little Rock, 2005.
- [11] Yin ling Cheung. FP-tree/FP-growth: Mining large itemsets using FP-tree algorithm. Technical Report [www.cse.cuhk.edu.hk/~kdd/freq/Nmost/ftp.zip](http://www.cse.cuhk.edu.hk/~kdd/freq/Nmost/ftp.zip), Chinese University of Hong Kong, 2002.