

HO CHI MINH UNIVERSITY OF TECHNOLOGY



## PARALLEL PROCESSING

## ASSIGNMENT REPORT

Group member

- Trần Đức Trung - 1552402
- Lê Hoàng Hiệp - 1450222

## Contents























I.	Introduction .....	2
1.	Association Rules .....	2
2.	OpenMP .....	3
II.	The Apriori algorithm .....	3
1.	How it works? .....	4
2.	Parallelizing Apriori algorithm using OpenMP .....	4
III.	Implementation .....	5
IV.	Speedup .....	5

# I. INTRODUCTION

## 1. Association Rules

**Association rule learning** is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. This rule-based approach also generates new rules as it analyzes more data. The ultimate goal, assuming a large enough dataset, is to help a machine mimic the human brain's feature extraction and abstract association capabilities from new uncategorized data.

The concept of **Association rule mining** was originated from the market basket analysis. It considers a transactional database  $D$  which consists of the transactional records of the customers. Each transaction  $T$  consists of the items purchased by the customers in one visit of the super market. The items are the subset of the set of whole items  $I$  in the super market we are considering for analysis. We represent  $I$  as the set. An itemset consists of some combination of items which occur together or a single item from  $I$ .

Transaction 1	   
Transaction 2	  
Transaction 3	 
Transaction 4	 
Transaction 5	   
Transaction 6	  
Transaction 7	 
Transaction 8	 

**Association rules analysis** is a technique to uncover how items are associated to each other. There are three interesting ways to measure this:

**Support:** This says how frequently the itemset appears in the dataset.

$$\text{Support} \{\text{🍎}\} = \frac{4}{8}$$

**Confidence:** This says how likely itemset Y is purchased when itemset X is purchased, expressed as  $\{X \rightarrow Y\}$ .

$$\text{Confidence} \{\text{🍎} \rightarrow \text{🍺}\} = \frac{\text{Support} \{\text{🍎, 🍺}\}}{\text{Support} \{\text{🍎}\}}$$

**Lift:** This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.

$$\text{Lift} \{\text{🍎} \rightarrow \text{🍺}\} = \frac{\text{Support} \{\text{🍎, 🍺}\}}{\text{Support} \{\text{🍎}\} \times \text{Support} \{\text{🍺}\}}$$

**Association rules** are usually required to satisfy two constraints *minimum support* and a *minimum confidence* at the same time (bot of them is defined by user). **Association rule** generation is usually split up into two separate steps:

Step 1: A *minimum support* threshold is applied to find all frequent itemset in a database.

Step 2: A *minimum confidence* constraint is applied to these frequent itemset in order to form rules.

## 2. OpenMP

**OpenMP** (Open Multi Processing) is an application program interface (API) that supports multi-platform shared memory multi-processing programming in C/C++ and Fortran on many architectures. It consists of a set of compiler directives. **OpenMP** uses *Fork-Join Parallelism* to implement multi-threading. (See <http://www.openmp.org> for specifications)

## II. THE APRIORI ALGORITHM

Many algorithms have been proposed for generating association rules. One of well-known algorithms is **Apriori** algorithm.

The **Apriori** algorithm can reduce the number of itemset we need to examine. For simple, this algorithm says that if an itemset is infrequent, then all its subsets must also be infrequent. This means that if {beef} was found to be infrequent, we can expect {beef, coke} to be equally or even more infrequent. So in the list of popular itemset, we need not consider {beef, coke}, nor any other itemset configuration that contains beef.

**Apriori** algorithm is used to find links between itemset in a transaction database. For example, a transaction database keeps track of items purchased by customers. It would be useful to find out which items that are most purchase together.

## 1. How it works?

By using the **Apriori** algorithm, the number of itemset that have to be examined can be pruned, and the list of popular itemset can be obtained in these steps:

Step 1: Start with itemset containing just a single item.

Step 2: Determine the support for itemset. Keep the itemset that meet the minimum support threshold, and remove itemset that do not.

Step 3: Using the itemset kept from Step 2, generate all the possible itemset configurations

Step 4: Repeat Steps 2 & 3 until there are no new itemset.

## 2. Parallelizing Apriori algorithm using OpenMP

We use the fork-join concept of **OpenMP** for finding frequent k- itemset. Pseudocode for **Apriori** algorithm in parallel:

All the items in the database will be in candidate 1-itemset, C1. For each item in C1, the following procedure will be followed.

1. divide the database into 2 partitions.
2. select a minimum support count ,min\_sup.
3. /\* start of parallel code

```
#pragma omp par+allel
```

```
#pragma omp sections{
```

```
    omp section{
```

```
        //partition1
```

```
        find_count1(i)
```

```
    }
```

```
    omp section{
```

```
        //partition2
```

```
        find_count2(i)
```

```
    }
```

```
}
```

```
/* implicit barrier */
```

/\* The first section calculates the count of items in partition1 and second section calculate the count of items in partition 2 \*/.

4. Sum up the counts of each partition separately for each item[i] in C1 and if count[i] > min\_sup, place item[i] in frequent 1-itemset, L1.

5. Join L1 with itself to get C2 where the items are of the type (i, j).

6. Go to step 4 for finding count of items in C2.

7. Repeat this process until  $L_k = \Phi$ .

### III. IMPLEMENTATION

The system created uses **OpenMP** to parallelize the tasks of check the support of items combinations versus the database transaction. For ease to understand and development, the transaction database is represented as a C struct, and randomly generated at the run time. Item in the database are integers (like ID or SKU in real case). The number of transactions can be defined by user at the run time. The support threshold (minimum support) which is how many time itemset appear in the transaction database, is also a command line option and specified by user at the run time.

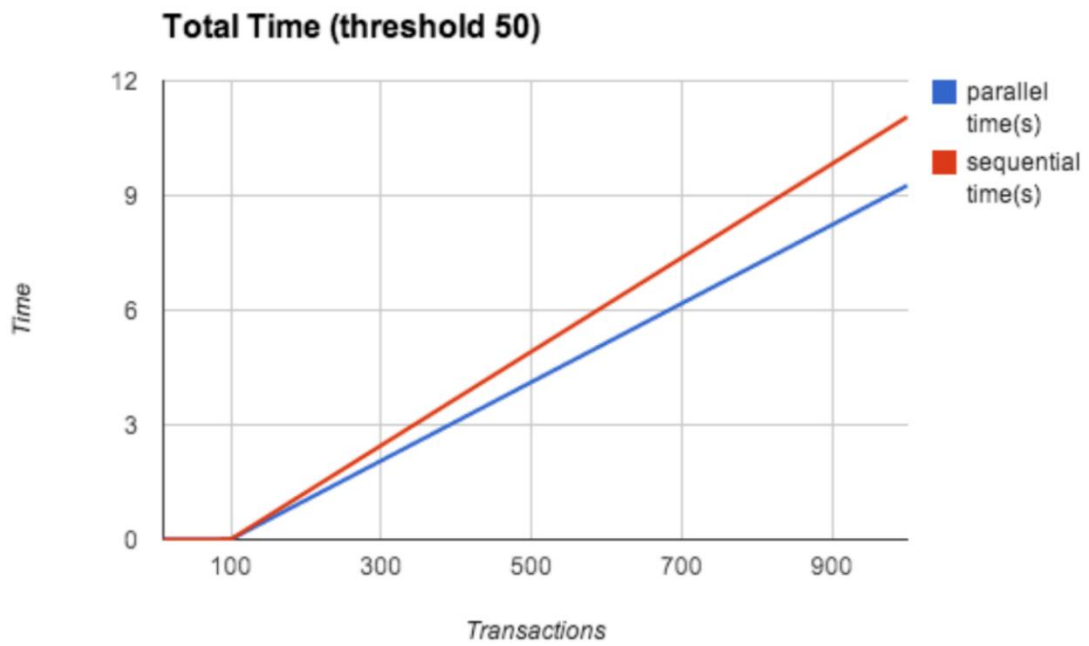
Once this program is running, user has to input the support threshold and the number of transactions. The program will print out all itemset and itemset combination that meet the defined threshold.

To run this program, type this command in the terminal:

```
./run.sh <min_sup> <number_transactions>
```

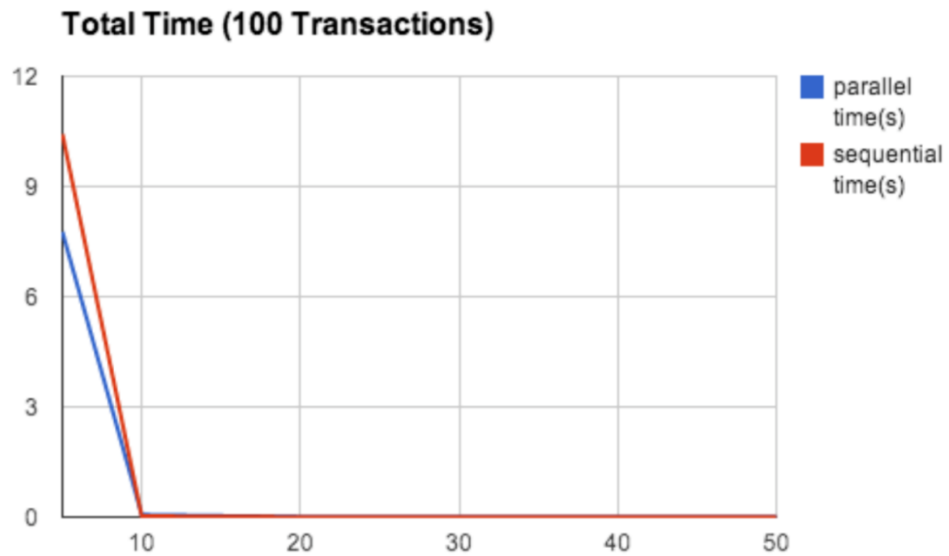
### IV. SPEEDUP

Time taken to run this program varies greatly depending on how many rounds the algorithm must do to find largest combinations that meet the minimum support threshold. We used an average of 5 runs where the maximum number of runs must execute.



X-axis: Total number of transactions.

Y-axis: Total run time in seconds.



X-axis: Support threshold.

Y-axis: Total run time in seconds.