Apriori-based Algorithm for Data Mining Using GPU Parallel Computing

Sheila Alemany[1], David Venta[2], Alex Dubuisson[3], and Amir Zarei[4]
School of Computing and Information Sciences, Florida International University
{salem010[1], dvent014[2], adubu002[3], azare004[4]}@fiu.edu

Abstract — There are numerous amounts of current data mining algorithms that properly evaluate and retrieve otherwise unknown information. However, as data doubles every single year, the complexity of the current implementations are reduced. In order for the complexity to improve as big data increases, parallelism is imperative. The Apriori algorithm is a typical, efficient algorithm for learning association rule mining. The main aim of this paper is to reduce the complexity of the Apriori algorithm using GPU parallel computing.

Keywords — *apriori algorithm; data mining; parallel computing*

I. INTRODUCTION

In the rapidly-growing world of technology, data mining is a relatively recent concept used to analyze data. Data which can provide information based on patterns, associations, or relationships [1]. This information can be converted into knowledge about past or future trends that can be of great assistance to many applications and companies. For the purposes of this paper, we will be focusing on a learning association rule mining algorithm called Apriori. Its intention consists of detecting the most frequent combinations of the different attributes in the data. It does so by using a "bottom up" approach, extending each of the frequent subsets one item at a time [2]. It begins with a group of candidate subsets of cardinality one, then traverses through the database, counts the number of occurrences of each itemset and eliminates the sets with the more infrequent sub patterns based on the minimum support threshold. This is repeated until we reach a group of cardinality sets where there are no occurrences in the database, implying that the group of cardinality sets from the previous iteration are the most frequent subsets.

Since this algorithm must iterate through the database multiple times, it is clear to see how as the quantity of data increases so does the complexity of the algorithm. The power of parallel processing has allowed for faster, more efficient analyzing by reducing the time complexity as much as to logarithmic time. We will be using NVIDIA's CUDA, a parallel computing platform, to implement the Apriori algorithm. Therefore, CUDA will allow us to successfully program multithreaded many core GPUs, as is already implemented throughout industry and academia to achieve significant speedups [3]. Overall, as one of the first research works that employs this algorithm in using NVIDIA's CUDA, we present an original parallel computing implementation taking advantage of the multithreaded many core GPU.

II. RELATED WORK

Parallel processing is not unknown to the Apriori algorithm. [4] implemented three different approaches. The Count Distribution approach which simply horizontally partitioned the database by transaction, where each partition belongs to its own process and each returns the local counts for all candidate itemsets. All counts are then summed up across all processes so that frequent itemsets can be identified. The Data Distribution approach which partitions both the database and the candidate sets. Each candidate set now has its own process in addition to the horizontal partition of the database. All processes then interchange database partitions during each iteration to properly assign the counts to each itemset. Lastly, the Candidate Distribution method which assigns each candidate itemset its own process and selectively replicates instead of exchanging as done in the Data Distribution approach. This was done in order for each process to proceed independently which allows for better performance and scalability compared to the first two approaches [4].

Another implementation used in [5] uses similar approach with a master/slave scheme where it partitions the number of transactions however, the master processor collects the unique items and divides them among the slave processors who compute the count and remove the items that do not meet the minimum support threshold. Once all the slave processors complete their computations, the results are sent to the master where it computes the next group of itemsets with an incremented cardinality and repeats the process again. In this implementation, there is a local hash table at the master which is accessed by each slave process.

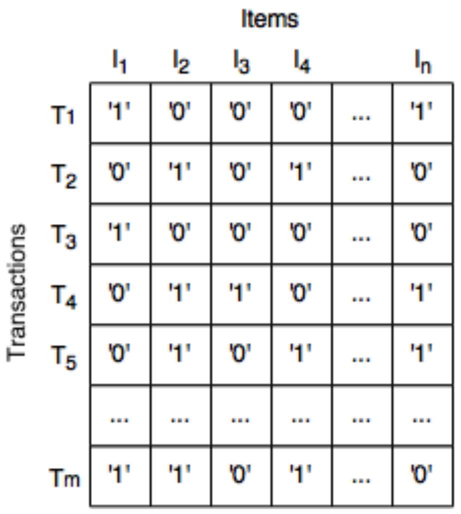
III. PROCESS

The Apriori algorithm's intention is to locate the subsets which are most common in a database where:

- An ***item*** (I_1, I_2, …, I_{n-1}, I_n) is defined as a subset of size one from the items in the given database.
- A ***transaction,*** T, is an entry in the given database that contains a group of itemsets where T = {I_1, I_2, …, I_n}.
- The ***database***, D, is defined as a set of all transactions where D = {T_1, T_2, …, T_m }. For the sake of quick access and less memory consumption of itemsets per transaction, the value of the itemset is stored as the index in the database and the presence of an itemset per transaction is represented by the character '1' and '0' otherwise.
- The ***minimum support threshold*** is defined as the amount of the times the itemset must appear in order to qualify as a frequently occurring relationship.
- The ***candidate subsets of cardinality size*** are subsets that have *k* amount of itemsets beginning at one in the first iterations possibly reaching to the total amount of itemsets, *n,* denoted as C = {C_1, C_2, …, C_n}.
- The ***frequency table*** contains all the candidate subsets of cardinality size and the total occurrence count.

(i) Step 1: Configuration

Our approach begins with counting the total number of transactions, *m*, and the maximum value of the items, *n*, in the data to properly set up our database on the CPU. Once the database is setup, we generate the first group of candidate subsets of cardinality one and copy the database and the frequency table to the GPU.



Items

Transactions

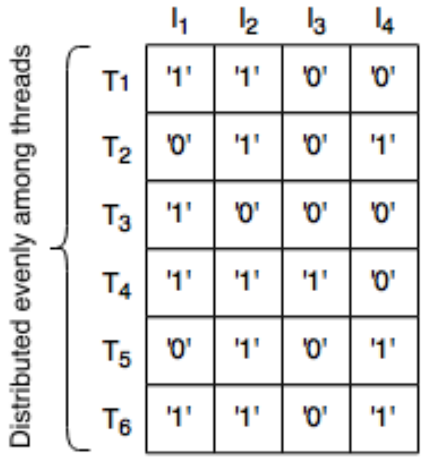
	I_1	I_2	I_3	I_4		I_n
T_1	'1'	'0'	'0'	'0'	...	'1'
T_2	'0'	'1'	'0'	'1'	...	'0'
T_3	'1'	'0'	'0'	'0'	...	'0'
T_4	'0'	'1'	'1'	'0'	...	'1'
T_5	'0'	'1'	'0'	'1'	...	'1'
	...	...	...	...	...	...
T_m	'1'	'1'	'0'	'1'	...	'0'

Figure 1: Database Configuration

In Figure 1 above, the database is shown as a two-dimensional array for clearer demonstration. However, both the frequency table and the database are setup in a one-dimensional array in order to copy the data to the GPU faster. This step only occurs once in our implementation.

(ii) Step 2: Parallelization

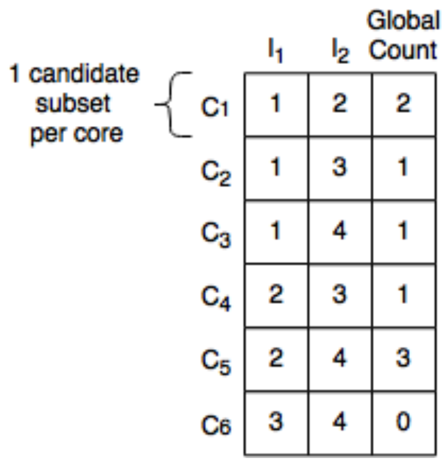
In our implementation, the cores are distributed among the individual candidate subsets of any cardinality size, *k*. Within each core, the transactions are divided among the 1024 threads available. Each group of transactions belong to one process or thread which checks the indices in the database of the item(s) in the candidate subset. Once the processes return the occurrences, they are summed up in parallel as well for the global count. This was done in order for the summation to occur in logarithmic time, while the threads check the occurrence of an item in constant time. A great improvement relative to a linear time complexity in both searching and adding in a non-parallel implementation.


Distributed evenly among threads

	I_1	I_2	I_3	I_4
T_1	'1'	'1'	'0'	'0'
T_2	'0'	'1'	'0'	'1'
T_3	'1'	'0'	'0'	'0'
T_4	'1'	'1'	'1'	'0'
T_5	'0'	'1'	'0'	'1'
T_6	'1'	'1'	'0'	'1'

Figure 2: Small Sample Database

Figure 2 above shows a sample database with four unique elements and six transactions. Figure 3 below shows a sample frequency table with a cardinality of size two. Each core will have six processes, where each process will check the indices of I_1 and I_2 in the database. These threads will return their results to another array in the


1 candidate subset per core

	I_1	I_2	Global Count
C_1	1	2	2
C_2	1	3	1
C_3	1	4	1
C_4	2	3	1
C_5	2	4	3
C_6	3	4	0

Figure 3: Sample Frequency Table with Cardinality of k = 2

GPU, not shown, and the values in this array will be added by another set of threads. The final result will be stored in the third column of the frequency table for the global count.

(iii) Step 3: Item Reduction

This count is compared with the minimum support threshold to determine which subsets must be eliminated. Once the invalid candidate subsets are identified, our algorithm:

1. Finds the remaining unique elements
2. Creates all possible sets from these elements of incremented cardinality
3. Current frequency table is saved as C_{n-1}
4. Creates a new frequency table, C_n, with the new candidate subsets
5. Jumps back to Step 2

The aforementioned steps are repeated until no candidate subsets of cardinality *k* meet the minimum support threshold. Implying that the sets of size *k-1* saved in the previous, C_{n-1}, frequency table are the most frequent itemsets in the database.

IV. RESULTS

To test our Apriori-based algorithm implementation, the dataset we used was found in [6], which contained an anonymized retail market basket data from an anonymous Belgian retail store. The file name used was ‘retail.dat’ and it contained a total of 16,469 items and 88,162 transactions. We defined our minimum support threshold to be 8000 as it is approximately 10 percent of all transactions. Our implementation returned six sets of cardinality two that met our minimum support requirements defined in a total time of 8348.41700 ms. In comparison, the Apriori implementation without parallelism implemented the same six set of cardinality two that met the same minimum support in a total time of 14066.482000 ms.

V. CONCLUSIONS

The Apriori-based algorithm for data mining proposed in this paper has shown to be an optimum approach that takes advantage of NVIDIA’s CUDA multithreaded many core GPU to reduce the time complexity and reduce the effects of big data on the Apriori algorithm. Furthermore, our configuration step also contributed to the time complexity greatly as it allowed each thread to check for the item in constant time. Overall, parallelism has shown a significant improvement in this learning association rule mining algorithm. Big data is going to increase dramatically as time passes, therefore parallelism is the going to be fundamental in future data mining.

VI. REFERENCES

[1] “Data Mining: What Is Data Mining?” Data Mining: What Is Data Mining? N.p., n.d. Web. 21 Nov. 2016.

[2] Suryawanshi, Sujata, et al. "Apriori Algorithm Using Data Mining.”

[3] Luebke, David. "CUDA: Scalable parallel programming for high-performance scientific computing." *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, 2008.

[4] Li, Jianwei, et al. "Parallel data mining algorithms for association rules and clustering." International Conference on Management of Data. 2008.

[5] Tarhini, Ali. "Parallel Apriori Algorithm for Frequent Pattern Mining." Ali Tarhini. N.p., 26 Feb. 2011. Web. 05 Dec. 2016.

[6] "Frequent Itemset Mining Implementations Repository." Frequent Itemset Mining Implementations Repository. N.p., n.d. Web. 02 Dec. 2016.

[7] Xiao, Han. "Towards parallel and distributed computing in large-scale data mining: A survey." Technical University of Munich, Tech. Rep (2010).

[8] Sanders, Jason, and Edward Kandrot. CUDA by Example: An Introduction to General-purpose GPU Programming. Upper Saddle River, NJ: Addison-Wesley, 2011. Print.