IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-6 E-ISSN: 2347-2693

An Effective and Optimized Approach to Association Rule Mining using GPGPU

Milind Kamath[1*], Ankit Katariya[2], Gaurav Bhokare[3]

[1*]Computer Engineering, PES Modern College of Engineering, Pune, India
[2]Computer Engineering, PES Modern College of Engineering/ SavitriBai Phule Pune University, Pune, India
[3]Computer Engineering, PES Modern College of Engineering/ SavitriBai Phule Pune University, Pune, India

Corresponding Author: milindkamath10@gmail.com, Tel.: +91-9763335324

Available online at: www.ijcseonline.org

Received: 22/May/2017, Revised: 04/Jun/2017, Accepted: 26/Jun/2017, Published: 30/Jun/2017

Abstract— Frequent Pattern Growth (FP-Growth) is a data mining technique, FP-growth algorithm introduced frequent pattern tree (FP-tree), stored as frequent item-sets in a compressed way. It overcomes drawback of candidate generation approach of multiple database scan but at the same time the transaction identifiers can be quite long taking substantial memory space and computation time. An optimised data structure viz. the Multi-Path Graph is used to improve the utilization and increase the efficiency of data mining techniques. Here we will be using graph as a data structure for storing frequent patterns in the memory. The graph structure will help to mine these frequent patterns without constructing FP-trees. However FP-Growth and MP-Graph fail to process extremely vast data-sets optimally. So we will be attempting to compare FP-Growth with MP-Graph as per its efficiency and memory utilization capability using parallelization techniques. We will try to achieve parallelization using CUDA, and bring forth a comparison of both the mining techniques.

Keywords—Associative rule mining , heterogenous parallel programming , CUDA , frequent pattern mining

I. INTRODUCTION

Association Rule Mining is a method of extracting data by uncovering interesting relations amongst variables in large datasets. Frequent item-set mining (FIM) is a tool use to discover these frequently co-occurring items. We came across a proposed novel approach for storing frequent patterns in the compact form to save the memory space and improve efficiency of the frequent pattern mining on the large datasets. The proposed novel structure is a very close-packed graph structure for store frequent patterns in the memory. It also reduces the overhead of constructing extra data structures for mining frequent pattern. Although to handle input data set consisting of millions of rows takes quite a long processing time. As data size keeps on increasing, input data processing cost causes a significant bottle-neck and this algorithm becomes expensive. We propose an approach to use CUDA and parallelize fraction of the target algorithm which focuses to reduce time required for input data processing. This gives an independent and cost effective method to solve the computation cost overhead caused by increasing size of datasets.

In this paper we present an algorithm to optimize input processing of proposed Multi-Path Algorithm using CUDA. We propose a sound mathematical analysis and solid figures to depict the speed up achieved by the optimized code.

II. RELATED WORK

Association rule mining algorithms can be classified based on candidate generation and test approach like in Apriori [1] [2], frequent pattern growth based algorithms such as FP-growth, CT-PRO, CP-TREE and algorithms that use vertical data format like Eclat. FP-growth [3] resolves the problems of huge candidate set generation, the time complexity and the multiple data scans for Apriori like algorithms. FP-growth reduces the number of scans from maximum length of frequent patterns to two scans. But the mining complexity in such approaches depends on the number of conditional trees constructed and the traversal cost associated [3] discusses the way of removing candidate generation techniques by proposing the FP-Tree structure where the database is first compressed in a smaller data structure reducing the cost of scanning the database repeatedly. It then uses the method of pattern fragment growth to reduce the creation of large amounts of candidate sets. It then divides the database into finer tasks to mine patterns within those databases. The efficient structure for trees did increase efficiency of FP-growth but when a large dataset is considered the number of conditional trees are high. Various studies based on this approach tried to make existing FP-tree structure more compact and improve mining performance. Parallel FP-Growth [4] implements the FP-growth mining on distributed architecture around MapReduce [4] discusses the need for increasing computation of Frequent Item-set mining techniques by using distributed computing. They propose Fp-

Tree technique for mining frequent items. Using distributed computing, each node will be working on its own chunk of database independent of other nodes. It is said to be promising in search engines for the support of query recommendation. The use of multiple systems increases the cost of such approaches tremendously. MP-Graph [5] developed a compact structure and the creation of conditional trees for mining using a graph like structure to attain the same [5-9] describes the limitation of creating redundant nodes while generating FP-tree for association mining techniques. They have proposed a novel structure called Multi-Path Graph as the name suggests.

III. SCOPE

Today's world is the world of data, data that is saved in a gargantuan amount of size. These data are being saved since two three centuries in the form of papers and now everything is digitised. Saving of huge data as databases isn't enough. Mining of such data for analysing and prediction is also necessary. Since the database is huge, it takes a large amount of time to analyse and further process it. Hence, the need for parallelising the computation of such huge databases for analysis will be of prime help in the long run.

IV. MP-GRAPH ALGORITHM

Module 1: This module is basically the crunching of huge datasets to get frequency of unique items for further graph creation and frequent pattern generation. The first step for module 1 is to read the dataset in the form of file and get the total count of transactions. The second step is to calculate the frequency of unique items in those transactions and save in the two arrays. These arrays are saved in descending order of frequencies calculated and pruned according to the support count. Using these steps we generate a Global Frequent Header Table GFHTable consisting of itemID and its frequency count, pointer to its consequent graph node and pointer to the next node.

Module 2: This module is creation of graph structure by using GFHTable generated in module 1. The pointer from Table to the graph node is used to initialise the node, and its subsequent structure within the node. The graph node consists of a sub structure which contains a parentreflist. This parentreflist is used to store the list of parents that the particular node can possibly have. This parentreflist is within itself a structure which is used to show the relation between the graph node and identify its parent. The parentreflist contains two fields, one being the parentrefnode which identifies the parent to the graph node and the other being the transaction bitmap which shows the presence of the parent child relation in terms of bitmap.

Module 3: This module uses bottom up approach to traverse the nodes we created in module 1. Frequent patterns are generated by traversing the graph for a particular item and going through the parentreflist till it becomes null. Conditional patterns are generated by using the bitmaps. The conditional patterns are pruned according to the confidence and combinations are found to obtain the frequent pattern sets.

Our Proposition: As we can see that the first module is the one in which huge processing takes place which consume more time, we intend to reduce its time by introducing parallel programming technique. This will substantially reduce the time complexity of the first module and reduce the overall time as well. We try to achieve parallelism by using CUDA programming.

V. PROPOSED ALGORITHM

A. Assumptions:

- Input to the program is given in form of a file.
- Every line in the file is a transaction.
- Transaction consist of item sets.
- Item sets are represented in integer format.
- File size should be minimum 3 lakh or more to achieve substantial speed up.
- Program requires proper sdk which supports nvcc compiler.

B. Algorithm Details:

Aim of our algorithm is to improve time efficiency of the application and achieve a considerable speed up factor. As discussed above, we intend to improve speed in the first module itself and hence we will walk through the algorithm for our first module only.

Algorithm:
On CPU:

1) For each transaction t in file:
2) For each item i in transaction:
3) Append i to the trans_arr
4) For each item i in trans_arr:
5) Copy i from trans_arr to trans_vect //2D array to 1D array
6) Sort trans_vect in descending order
7) Create item_arr, freq_arr of size trans_vect
8) Compute frequency count of item i in trans_vect by reduction, store in freq_arr
9) Pass unique items i from trans_vect to item_arr
10) Shrink item_arr and freq_arr to prune left over zeros.

11) Rearrange trans_arr according to the to the freq_arr values.

On Gpu :(enable parallelism)

Considering T threads provided by gpu and for N number of elements:-

1) For each transaction t in file:
2) For each item i in transaction:
3) Append i to the trans_arr
4) For each item i in trans_arr by thread id N/T*i:
5) Copy i from trans_arr to trans_vect //2D array to 1D array
6) Sort trans_vect in descending order using T threads
7) Create item_arr, freq_arr of size trans_vect
8) For i in trans_vect:
9) Count unique occurrence of i using T threads i.e N/T*i partitions
10) For i in trans_vect:
11) Count unique occurrence of i and update item_arr
12) Shrink item_arr and freq_arr to prune left over zeros using T threads.
13) Rearrange trans_arr according to the to the f freq_arr values using Threads.

VI. MATHEMATICAL ANALYSIS

For serial code: (on CPU only)
Time complexity:

O (n+m*p) + O (mlogm) + O (m^2) ------ (1)

Similarly for parallel code (on GPU):

Time complexity:

The second and the third module aren't optimised and hence their time complexity will be O (mlogm) + O (m^2) ------ (2)

For the first part we have serial complexity as O (n*+m*p) where n is the transactions in the DB.
We have launched threads in the number of transactions which means we have launched x threads which are equal to the number of number of transactions.

I.e. x=n ----- (3)

Thus we have parallelised the first part of GFHTable generation.

Hence its time complexity is:

O (n/x + m*p)
And from (3) we get:

O (1+m*p) ----- (4)

From (2) and (3) we get time complexity as O (1+m*p) + O (m logm) + O (m^2) ----- (5)

Since the main concern for parallel code is to find the optimisation or the speed up,
Hence we will find the speedup

We have to first find out how fast parallel code is than serial code.

Thus
execution_time_for_serial_code/execution_time_for_parallel_code = 1+N/100

N is percentage faster

From our analyses we have for 60 lack transactions

execution_time_for_serial_code = T_s Ms
execution_time_for_parallel_code = T_p Ms

Thus
$T_s/T_p = 1 + N/100$

$N/100 = T_s/T_p - 1$ % faster

We have parallelised approximately 1/3rd of our code
I.e. 33.33333%

Using Amdahl's law:

Overall speedup if we make 33.33333% of our code T_s/T_p - 1 % faster
$F = 0.333$ $S= (T_s/T_p - 1) /100$

Speedup = 1/ ((1-F) +F/S)
= 1 / ((1-0.333) + 0.333/(T_s/T_p -1) /100

Using the values of T_s and T_p , we will be able to calculate the exact speedup of our program.

VII. CONCLUSION

In this paper we discussed the implementation of fractional parallelization of Association Rule Mining algorithm MP-Graph using CUDA. The outcome will show that GPU outperforms CPU and will conclude that in general using GPU based implementations we can process huge data files much faster than processing same files on CPU based implementations. The algorithm resolves a major issue of the required computational time which will be reduced significantly by our parallel approach. Combining the space -

time efficiency achieved by parallel MP-Graph algorithm we will achieve to produce a balanced algorithm to generate frequent patterns.

REFERENCES

[1]. R Agrawal, T Imielinski and A Swami, "*Mining association rules between sets of items in large databases*" In the proceedings of the SIGMOD '93 ACM SIGMOD international conference on Management of data Pages 207-216

[2]. J Han, J Pei, Y Yin. and R Mao, "*Mining Frequent Patterns without Candidate Generation*" In the proceedings of SIGMOD '00 of the 2000 ACM SIGMOD international conference on Management of Pages 1-12

[3]. H Li, Y Wang, D Zhang. and M Zhang, "*PFP: Parallel FP Growth for Query Recommendation*" In the proceedings on the ACM conference on Recommender system, pp 107-114 ACM (2008)

[4]. R.V. Mane, V.R. Ghorpade, "*Use of Constraints in Pattern Mining: A Survey*", International Journal of Computer Sciences and Engineering, Vol.4, Issue.11, pp.95-99, 2016.

[5]. M. Dhivya, D. Ragupathi, V.R. Kumar, "*Hadoop Mapreduce Outline in Big Figures Analytics*", International Journal of Computer Sciences and Engineering, Vol.2, Issue.9, pp.100-104, 2014.

[6]. V. Jain, "*Frequent Navigation Pattern Mining from Web usage data*", International Journal of Scientific Research in Computer Science and Engineering, Vol.1, Issue.1, pp.47-51, 2013.

[7]. Nidhi Sethi and Pradeep Sharma, "*Mining Frequent Pattern from Large Dynamic Database Using Compacting Data Sets*", International Journal of Scientific Research in Computer Science and Engineering, Vol.1, Issue.3, pp.31-34, 2013.

[8]. Marie Fernandes , "*Data Mining: A Comparative Study of its Various Techniques and its Process*", International Journal of Scientific Research in Computer Science and Engineering, Vol.5, Issue.1, pp.19-23, 2017.

[9]. Jaswant Meena, Ashish Mandloi , "*Classification of Data Mining Techniques for Weather Prediction*", International Journal of Scientific Research in Computer Science and Engineering, Vol.4, Issue.1, pp.21-24, 2016.

[10]. Deepti Sharma and Vijay B. Aggarwal, "*Mapreduce- A Fabric Clustered Approach to Equilibrate the Load*", International Journal of Computer Sciences and Engineering, Vol.4, Issue.3, pp.116-123, 2016.

Authors Profile

Milind Madhav Kamath is a BE Final Year student in the Computer Department, Progressive Education Society's Modern College of Engineering, Pune. Will receive Bachelor in Computer Engineering in the year 2017 from Savitribai Phule Pune University, Pune, and Maharashtra, India. Interests in c/c++, java, cuda, android hacking.

Gaurav Ajay Bhokare is a BE Final Year student in the Computer Department, Progressive Education Society's Modern College Of Engineering, Pune. Will receive Bachelor in Computer Engineering in the year 2017 from Savitribai Phule Pune University, Pune, Maharashtra, India. Interests in c, cuda, OpenGL.

Ankit Paras Katariya is a BE Final Year student in the Computer Department, Progressive Education Society's Modern College Of Engineering, Pune. Will receive Bachelor in Computer Engineering in the year 2017 from Savitribai Phule Pune University, Pune, Maharashtra, India. Interests in c/c++,Algorithm Design.