Parallel FIM Approach on GPU using OpenCL

Sarika S. Kadam
Research Scholer
Department of Computer Engineering
Pimpri Chinchwad College of Engineering
Pune, India.
Email: sarikaengg.patil3@gmail.com

Prof. Sudarshan S.Deshmukh
Department of Computer Engineering
Pimpri Chinchwad College of Engineering
Pune, India.
Email:deshmukh.sudarshan@gmail.com

***Abstract*—** In this paper, we describe GPU-Eclat algorithm, a GPU (General Purpose Graphics Processing Unit) enhanced implementation of Frequent Item set Mining (FIM). The frequent itemsets are extracted from a transactional database as it is a essential assignment in data mining field because of its broad applications in mining association rules, time series, correlations etc. The Eclat approach is the typically generate-and-check approach to obtain frequent itemsets from a database with a given minimum support threshold value. OpenCL is a platform independent Open Computing Language for GPU computation. We tested our implementation with an Radeon Dual graphic processor and determine up to 68X speedup as compared with sequential Eclat algorithm on a CPU. In order to map the Eclat algorithm onto the SIMD (Single Instruction Multiple Data) execution model, an array data structure is used to represent the input database and standard dataset is converted to the vertical data layout. In our implementation, we perform a parallelized version of the candidate generation and support counting phases on the GPU. Experimental results show that GPU-Eclat consistently outperforms CPU-based Eclat implementations. Our results reveal the potential for GPGPUs in speeding up data mining algorithms.

***Keywords*—** *Eclat, frequent Itemset mining ,GPU, SIMD, OpenCL.,Parallel computing*

*****

I. INTRODUCTION

An incredible growth of data needs to be processed in business applications and scientific research areas. Extracting information from large amount of data is necessary in making correct and effective decisions. Different methods have been developed to determine the characteristics and inter-relationships of data. Discovery of common and interesting patterns from databases is an important goal of frequent itemset mining. Finding frequent item sets [16] in a set of transaction is a prevalent method for market basket analysis, which aims at finding symmetries in the shopping activities of client of super market, online shop etc. Association rule learning, classification, clustering, and regression , decision support, financial forecast, marketing policies, even medical diagnosis and many other applications are commonly need to mine data. Association rules describe how often items are purchased together. For example, an association rules "beer, chips (80%)" states that four out of five customers that bought beer also bought chips. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others.

The Frequent Itemset Mining (FIM) problem was introduced by Agrawal et al. [7, 8], as the first step to mine association rules in market basket data. Let I = {I1, I2, …., Im} be a set of m items, and T = {T1, T2, ….Tn} the transaction database, where Ti is a transaction containing a set of items from I. A k-itemset that consists of k items from I, is frequent if it occurs in T not less than s times, where s is a user-specified minimum support threshold. A FIM algorithm tests the database, possibly several times, and finds item-sets that occur in transactions equal or more frequently than a given minimum threshold. The frequency of items that are present in a transaction is called support. Apriori, Eclat and FP-Growth [4] these are the best known FIM algorithms.

A GPU, [1, 5, 6] is a coprocessor to process an image, for games and to support the CPU for graphics processing applications such as matrix multiplication and distributed computing projects including Folding@home and Seti@home.. There are thousands of computing units in a GPU; each unit is a simplified core of CPU. The number of GPU cores is more than that of CPU cores, so GPU is suitable for parallel computing. General-purpose computing on graphics processing units (GPGPU) was proposed to provide non-graphic computing capabilities to CPU. GPU is a high-performance computing device which does not only reduce the deployment cost but also saves on maintenance.

NVIDIA and ATI have been proposed as GPU programming language by CUDA and Stream respectively. Previous frameworks could only be used with the respective GPUs, e.g., CUDA could only be executed on NVIDIA's GPUs. The Khronos Group and many industry-leading companies created the OpenCL to deal with platform heterogeneity. OpenCL is an open and cross-platform parallel heterogeneous programming system which provides a uniform programming environment for developers to write efficient and portable codes in any system which is composed of different CPUs, GPUs, and other computing platforms and able to perform in different operating systems. The CPU and GPU can then communicate with each other and work together by applying the appropriate file for CPU and Kernel file for OpenCL on GPUs to perform parallel computation with GPU.

II. RELATED WORK

There has been much recent interest in implementing FIM algorithms. Best known FIM algorithms are Apriori [4, 8] & Eclat [4]. Apriori & Eclat iteratively generates K sized frequent item sets by joining frequent K-1 sized item sets. This step is called candidate generation. After generating each new set of candidates, algorithm scans the transaction database to count the no. of occurrences of each candidate. This step is called support counting. The primary difference between

Apriori & Eclat is the way they represent candidate & transaction data & the order that they scan the tree structure that stores the candidates.

A. Sequential Implementation:

There has been much recent interest in implementing FIM algorithms. Ferenc Bodon implemented Apriori using trie-based data structure & candidate hashing [10], Christian Borgelt implemented Apriori in his work [16] using recursion pruning. The Borgelt Gelat is capable of detecting dataset characteristics & automatically choosing the best corresponding data representation (including Tidset, Bitset, Diffset..).

Eclat [16] traverses the prefix tree in depth first order. It extends an item set prefix until it reaches the boundary between frequent & infrequent item sets and then backtracks to work on the next prefix. Eclat determines the support of an item set by constructing the list of identifiers of transactions that contain the item set. It does so by intersecting 2 lists of transaction identifiers of 2 item sets that differ only by one item & together form the item set currently processed.

B. Message Passing Parallel Implementation:

Ye et al. demonstrated a parallel Apriori Algorithm based on a revised Bodon implementation that achieved a 2x speedup with 8 processors [15]. Craus developed on MPI-based parallel Apriori algorithm that distributed the transaction among computing nodes [17] .Another trie-based MPI implementation based on Bodon's algorithm was developed by Ansari et al
[18].

C. GPU Related Implementation:

Fang developed a GPU implementation of Apriori [1]. In this case, two versions of their GPU implementation, one based on the "pure bitmap" representation & another based on the "trie-based bitmap" representation were described. In their approach the candidates & vertical transactions are coded into bitmaps & manipulated on the GPU. They used an NVIDIA Getforce GTX 280 GPU to test their algorithm. Their method achieved a speedup of 2x-10x as compared with a CPU-based serial Apriori implementation.

Fan Zhang developed GPApriori, a GPU implementation of frequent itemset mining(FIM)[2].In order to map Apriori algorithm onto the SIMD execution model ,"Static bitset" memory data structure have been designed to represent input database, which improves upon traditional approach of vertical data layout. Support counting is performed parallely on GPU. GPApriori performs better than CPU-based Apriori implementation.

Fan zhang Developed new parallel frequent itemset mining algorithm called "Frontier Expansion"[3]. High performance on a heterogeneous platform is achieved which consists of a shared memory multiprocessor and multiple GPU coprocessors. Frontier expansion is an improved data parallel algorithm derived from Eclat method. In this approach 4 NVIDIA Tesla GPUs are used to achieve 6-30x speedups relative to sequential Eclat implementation executed on a multicore CPU.

III. IMPLEMENTATION DETAILS

A.Eclat Algorithm

An Eclat is a depth-first search algorithm which refers set intersection. Vertical database layout is referred for illustration. i.e. all transactions are not listed explicitly but each item is stored together with its cover and uses the intersection based approach to compute the support of an itemset. The support of an itemset A can be easily calculated by cover's intersection of any two subsets Y, Z ⊆ A, such that Y U Z = A. Candidate generation of Eclat uses only the join step of Apriori , since the item sets necessary for the prune step are not available. The size of tidsets is one of main factors affecting the running time and memory usage of Eclat. The bigger tidsets are, the more time and memory are needed.Fig.2 shows representation of data: horizontal data layout and vertical data layout. And Fig.3 shows example which demonstrates working of Eclat algorithm.

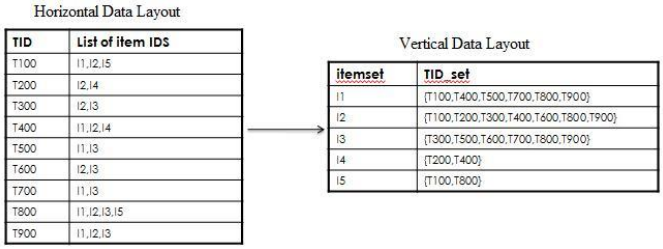

Fig. 1. Layouts for illustration of the data

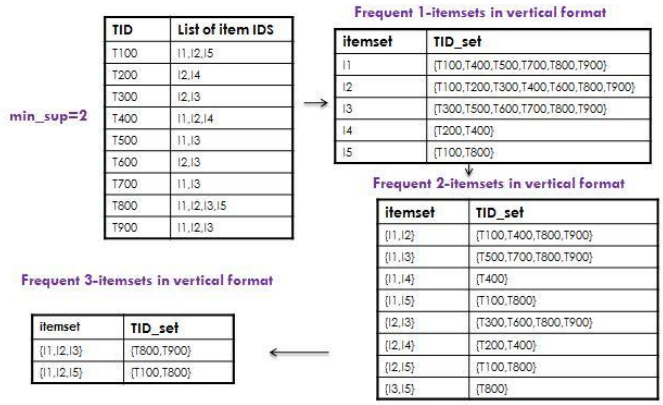

Fig.2 Illustrating example of Eclat algorithm

B.Projected Algorithm: GPU-Eclat

Input: a transaction database D and a minimum support threshold.
Output: a complete set of frequent itemsets.

---

1. Accept Standard dataset and minimum threshold support from user.
2. Generate vertical Tidlist by scanning D and store it on CPU Memory
3. Start the CL program to be executed by the GPU.
4. Allocate memory space in GPU for vertical Tidlist.
5. Store Tidlist of each item into GPU memory
6. Generate and verify first level frequent itemsets by counting and comparing support value of each tidlist with minimum threshold.
7. Allocate memory space in GPU for candidate itemsets
8. Perform launch kernel of clProg for candidate generation (on GPU)

a. Each Processing Element (PE) of GPU allocated for each item
b. for each item in vertical data layout
   i. Processing Element computes candidates by intersecting tidlists of previous level frequent itemsets for GPU threads to process.
   ii. Each candidate is given to Processing Element for support counting

9. Repeat the step 8 until all the same level candidate itemsets are done.
10. Allocate memory space in GPU to save the results
11. Perform launch kernel of clProg for support counting (on GPU)
    a. Each Processing Element (PE) of GPU allocated for a set of candidate itemsets (CIs)
    b. for each Candidate in CIs
       i. Processing Element computes the support of Candidate.
       ii. If support of Candidate is greater than or equal to minimum support threshold, then set it is frequent else set it is not frequent.
12. Move to next level candidate set generation and perform Steps 8-11 until all level Candidates are generated and verified with minimum threshold.
13. Verify time for GPU required finishing frequent itemset generation, retrieving the results to CPU memory.

C.Proposed Approach:

In this approach, we have implemented GPU-based FIM algorithm. GPUs are multi-threaded many-core processors on which, cores are virtualized, and GPU threads are executed in SIMD (Single Instruction, Multiple Data) and are managed by the hardware. Such a design simplifies GPU programming and improves program by making it scalable and portable. Apriori or Eclat implementation on the GPU [1],[2],[3]is a quite challengeable, an array data structure is used to represent transactions in this GPU-based Eclat FIM implementation. Specifically, the array is converted into the vector which stores the occurrences of items in transactions, and is efficient to be partitioned to SIMD processors. Vertical data layout is used to facilitate candidate generation and support counting operations in FIM, where support counting is the most time consuming component in the FIM algorithm.

The proposed system is divided into three modules: vertical data representation, candidate generation and support counting. The standard dataset is converted into vertical data layout .In first level by counting cardinality of tidlist of each item , support is counted and compared with minimum support threshold for generating frequent itemsets. Then for next level, intersections of tidlists of previous frequent itemsets are considered for candidate generation. For each level tidlists are allocated to processing element of GPU in parallel thus parallelization is utilized. And for next level candidate generation, previous level tidlists are considered for intersection, thus synchronization is utilized. And support counting operation at each level is performed in parallel by allocating resultant tidlist to each processing element of GPU. In this approach, Eclat algorithm is implemented on CPU as well as on GPU by considering following hardware and software configuration. After execution of both the cases, the execution time , total number of frequent itemsets generated & total number of transactions present in selected dataset are displayed .And generated frequent itemsets with corresponding support count is saved in file whichever user have specified as resultant storage file. The speed-up ratio is calculated from calculated CPU execution time and GPU execution time, as a performance metric for the approach.

III. RESULT ANALYSIS

A.Data set

The dataset is obtained from the FIMI (Frequent Itemset Mining Implementation) repository. The four standard datasets are used namely: Chess, Mushroom, Pumsb, Pumsb*.The dataset mushroom is a description of hypothetical samples corresponding to different species of mushrooms. This dataset consists of 8124 instances of 119 attributes which are derived from 24 species. It is a dataset containing long pattern with significant repetitions and overlaps between transactions, i.e., a dense dataset. The chess dataset is also a dense dataset containing of 3196 instances and 74 items.

The synthetic data generator takes as input the parameters: The average length of a transaction, the number of items in a dataset, the number of transaction in the dataset. The characteristics of datasets are given below:

TABLE 1. Experimental datasets

Dataset Name	#Items	Avg. Length	#Transactions
Chess	75	37	3,196
Mushroom	119	23	8,124
Pumsb-star	2088	50.5	49,046
Pumsb	2113	74	49,046

B. Proposed Eclat Result:

We evaluated the visualization performance of Eclat with different support thresholds, on the four Standard dataset taken from FIMI Repository namely, Chess, Mushroom, Pumsb & Pumsb* etc. The Speed up ratio is the performance parameter for whole system. Execution time for generation of frequent itemsets on inputs i.e. dataset and min. support threshold for sequential Eclat on CPU and parallel Eclat on GPU having a greater difference .GPU Eclat is faster than CPU Eclat algorithm as compared with parameter speed up ratio.

TABLE 2: Support vs Speed-up ratio for standard datasets

Speed-up ratio				
Support	Chess	Mushroom	Pumsb*	Pumsb
0.1	94.4	90.112	92.55	94.09
0.2	93.85	89.425	92.19	94.07
0.25	93.2	82.362	90.87	87.59
0.3	92.9	81.288	87.29	87.43
0.35	92.89	78.713	86.92	89.71
0.4	92.8	77.17	82.51	89.64
0.45	92.761	35.825	76.57	86.09
0.5	92.688	35.67	76.18	56.371

 ISSN: 2321-8169
3216 – 3221

0.55	90.127	35.19	47.62	36.85
0.6	82.188	32.79	38.46	37.76
0.65	78.993	32.45	38.17	43.642
0.7	76.985	33.692	37.34	37.3
0.75	75.207	33.02	36.66	20.2

The result is tested for four datasets namely, chess , Mushroom and Pumsb* and Pumsb for various support values.And CPU time as well as GPU time is noted for each reading. And then Speed up ratio is calculated for various support values.

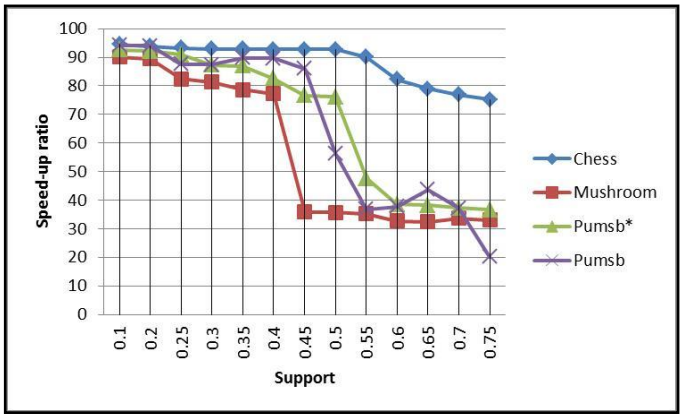

Fig. 3: Graph for Support Vs Speed-up ratio on Chess, Mushroom, Pumsb, Pumsb*

Mean speed up ratio for chess is calculated as 88.38, for mushroom as 56.75, for pumsb as 66.21 and for pumsb* as 67.95. By comparison, it is proved that Chess dataset is giving the better speedup than other three.

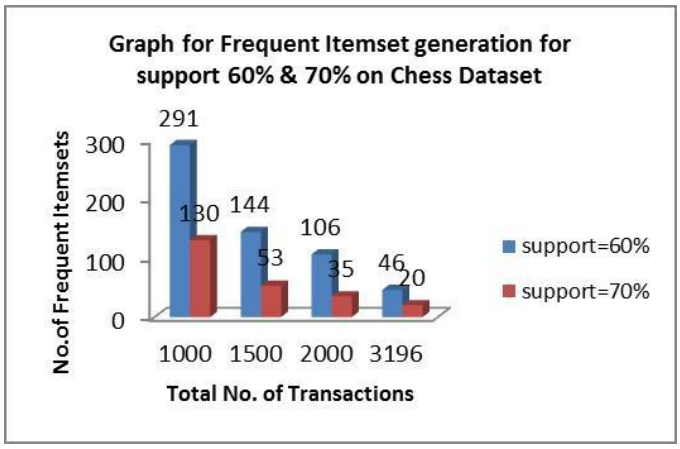

Fig. 4: Graph for F.I. generation on Chess dataset for 60% & 70% support

Above graph is plotted by considering total no. of transactions and no. of frequent itemsets generated. As the no. of transactions increases for both the support 60% & 70% on chess dataset, the numbers of frequent itemsets generated are increased as shown in graph.

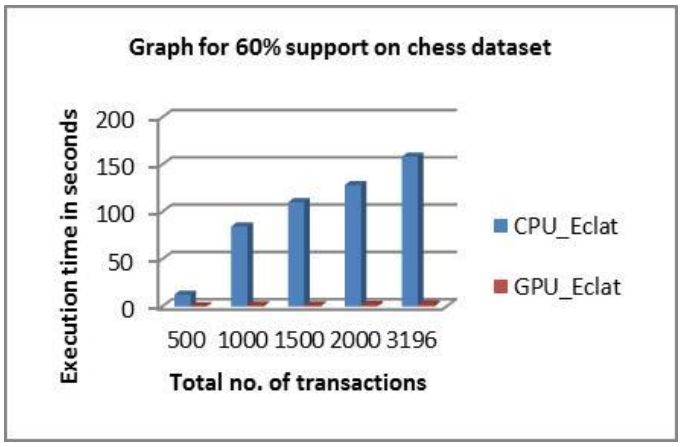

Fig. 5: Graph for 60% support on Chess dataset with varying no. of transactions

Graph is plotted by considering no. of transactions & Execution time .As transactions are increasing, execution time required for frequent itemset generation also increases.

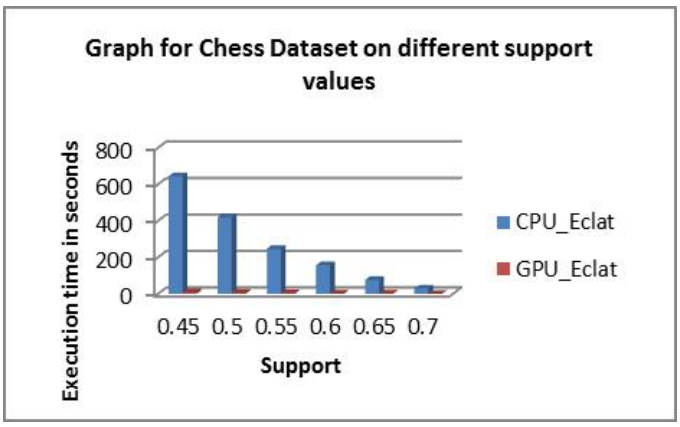

Fig. 6: Graph for Chess dataset on different support values

Above graph is plotted by considering support values & execution time for Chess dataset. It is observed that as the support value is increasing, the execution time is getting reduced .Because as the support value decreases, the number of frequent itemset generated are increasing, so execution time is larger in that case.

For each support value , when execution time is compared for frequent itemset generation it is always less for GPU device for same input (dataset and support), thus performance is improved when it is compared with CPU execution time.

ISSN: 2321-8169
3216 – 3221

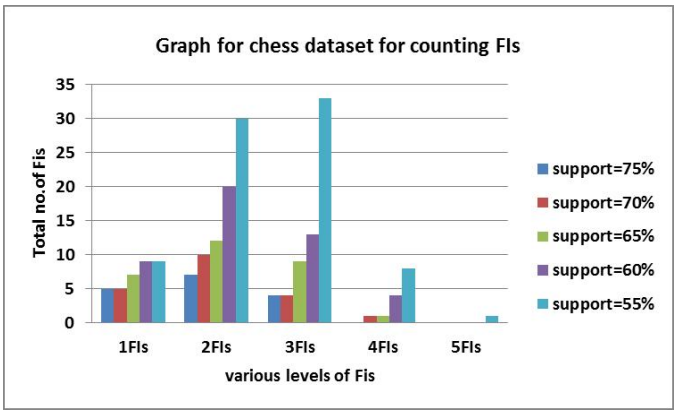

Fig.7: Graph for counting frequent Itemsets at five levels on Chess dataset

Above graph is plotted by considering various levels of frequent itemsets & total no. of frequent itemsets generated for Chess dataset on various support values. It is observed that for 55% support upto all five level frequent itemsets are generated, for 60%, 65% & 70% support upto 4 levels frequent itemsets are generated, for 75% support upto 3 levels frequent itemsets are generated. As the support value decreases, number of frequent itemsets generated is increased.

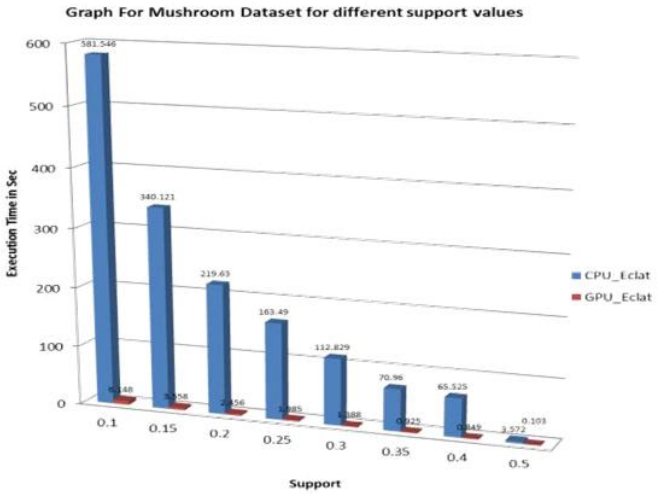

Fig.8: Graph for Mushroom Dataset for different support values

Above graph is plotted by considering support values & execution time for Mushroom dataset. It is observed that as the support value is increasing, the execution time is getting reduced. Thus support value is inversely proportional to execution time.

For each support value , when execution time is compared for frequent itemset generation it is always less for GPU device for same input (dataset and support), thus performance is improved when it is compared with CPU execution time.

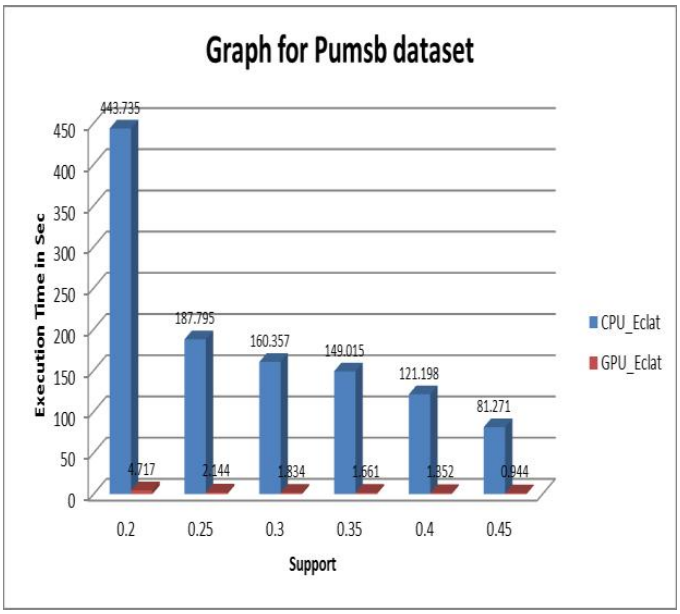

Fig. 9: Graph for Pumsb dataset for different support values

Above graph is plotted by considering support values & execution time for Pumsb dataset. It is observed that as the support value is increasing, the execution time is getting reduced. Thus support value is inversely proportional to execution time.

For each support value , when execution time is compared for frequent itemset generation it is always less for GPU device for same input (dataset and support), thus performance is improved when it is compared with CPU execution time.

It is observed that for support >=0.8, there are no any frequent itemsets are generated for Pumsb dataset.

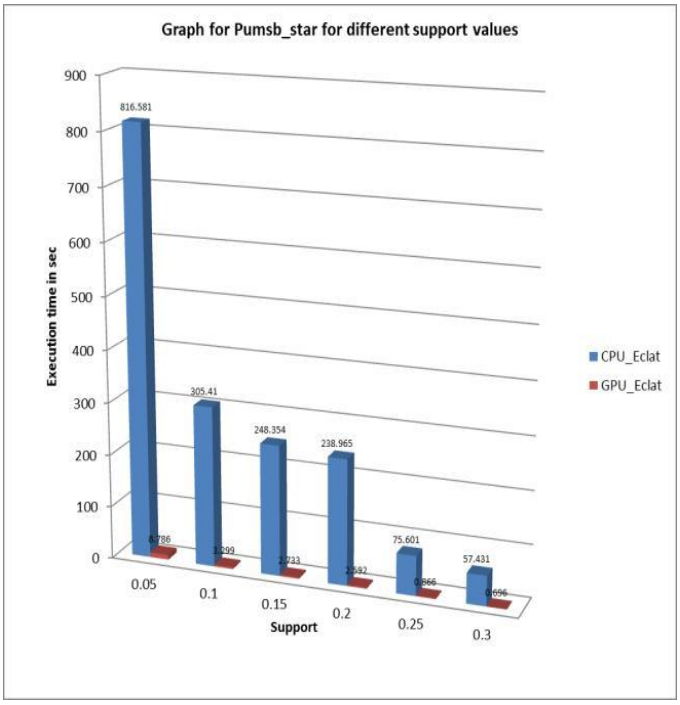

Fig.10: Graph for Pumsb_star dataset for different support

values

Above graph is plotted by considering support values & execution time for Pumsb-star dataset. It is observed that as the support value is increasing, the execution time is getting reduced. Thus support value is inversely proportional to execution time.

For each support value , when execution time is compared for frequent itemset generation it is always less for GPU device for same input (dataset and support), thus performance is improved when it is compared with CPU execution time.

It is observed that for support >=0.8, there are no any frequent itemsets are generated for Pumsb-star dataset.

IV. CONCLUSION AND FUTURE WORK

We have implemented GPU-based implementation of Eclat algorithm for frequent itemset mining .The implementation employ an array data structure to encode the transaction database on the GPU and exploit the GPU's SIMD parallelism for support counting. The implementation stores the itemsets in an array, and runs entirely on the GPU. i.e. vertical transaction lists are represented using vectors across multiple threads on a GPU. The evaluation results show that the GPU-implementation is up to 68x faster than CPU-based implementation of Eclat. Parallel Eclat algorithm using GPU is better than the Eclat CPU as the processing speed gets increased with efficient computation.

It is a challenging to develop a buffering mechanism between the GPU memory and the CPU memory which will reduce computation time. It is the challenge for accelerating the frequent Itemset mining on GPU with Dynamic Dataset.

REFERENCES

[1] Wenbin Fang, Mian Lu, QiongLuo, Xiangye Xiao Frequent Itemset Mining on Graphics processors [*Proceedings of the fifth International workshop on data management 2009*]

[2] Fan Zhang, Yan Zhang, Jason D. Bakos GPApriori: GPU – Accelerated Frequent Itemset Mining [*2011 IEEE International Conference on Cluster Computing*]

[3] Fan Zhang, Yan Zhang, Jason D. Bakos Accelerating frequent itemset mining on graphics processing units [*2013 Springer Science+ Business Media New York*]

[4] Pramod S., O.P.Vyas Survey on Frequent Item set Mining Algorithms [*International Journal of Computer Applications (0975-8887) vol. 1-No.15*]

[5] Jiayi Zhou, Kun-Ming Yu, Bin-Chang Wu Parallel frequent Patterns Mining Algorithm on GPU [*2010 IEEE National Science Council*]

[6] Che-Yu Lin, Kun-Ming Yu, Wen Ouyang, Jiayi Zhou An OpenCL Candidate Slicing Frequent Pattern Mining Algorithm on Graphic Processing Units [*2011 IEEE National Science Council*]

[7] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD, 1993*.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules.*VLDB*, 1994.

[9] Lamine M. Aouad, Nhien-An Le-Khac, and Tahar M. Kechadi. Distributed frequent itemsets mining in heterogeneous platforms. *Journal of Engineering, Computing and Architecture, 2007*.

[10] Ferenc Bodon. A fast apriori implementation.*FIMI*, 2003.

[11] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheafer, and Kevin Skadron . A performance study of general-purpose applications on graphics processors using cuda.[*Journal of parallel and Distributed Computing, 2008.*]

[12] http://fimi.cs.helsinki.fi/. *FIMI repository*.

[13] http://www.adrem.ua.ac.be/goethals/software/files/apriori.tgz. Apriori implementation from Bart Goethals.

[14] Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens. Scan primitives for gpu computing. *Graphics Hardware, 2007*.

[15] Yanbin Ye and Chia-Chu Chiang. A parallel apriori algorithm for frequent itemsets mining. *SERA, 2006*.

[16] Christian Borgelt Efficient Implementation of Apriori and Eclat [*Dept. of Knowledge Processing*]

[17] Craus M. A new parallel algorithm for the frequent itemset mining problem.[*International Symposium on parallel & distributed computing, 2008, ISPDC '08, PP 165-170*]

[18] Ansari G, Dastghai Gifard G. Distributed Frequent Itemset mining using trie data structure. [*Int J Computer Sci.35 (3): 377-381,2008*]