IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. VIII (Mar-Apr. 2014), PP 88-96
www.iosrjournals.org

Eclat Algorithm for FIM on CPU-GPU co-operative & parallel environment

Sarika S.Kadam[1] ,ME Scholar, Computer Engg. Dept.Sudarshan S.Deshmukh[2] ,Asst. Professor. Computer Engg.Dept.
[1,2], *PimpriChinchwad College of Engineering , Akurdi, Pune, Pune University.*

Abstract: *Extracting the frequent itemsets from a transactional database is a fundamental task in data mining field because of its broad applications in mining association rules, time series, correlations etc. The Apriori or Eclat approaches are the commonly used generate-and-check approach to obtain frequent itemsets from a database with a given threshold value.*

Implementations take advantage of the GPU's massively multi-threaded SIMD (Single Instruction, Multiple Data) architecture which will employ a bitmap data structure to represent vertical transaction list ,to exploit the GPU's SIMD parallelism , and to perform the support counting operation. The implementation runs entirely on the GPU and eliminates intermediate data transfer between the GPU memory and the CPU memory, which can reduce computation time and improve overall performance .OpenCL is a platform independent Open Computing Language for GPU computation. Thus, the aim of our approach is to develop efficient parallel new advanced Eclat strategy of Frequent Itemset Mining that utilize new-generation graphics processing units (GPUs) to speed-up the process.

Keywords: *Eclat, Apriori, frequent Itemset,GPU, Bitmap, SIMD, OpenCL.*

I. INTRODUCTION

Frequent itemset mining (FIM)[4] aims at finding common and interesting patterns from databases. Finding frequent item sets[16] in a set of transaction is a popular method for so-called market basket analysis, which aims at finding regularities in the shopping behavior of customer of super market, mail- order companies, online shop etc. Identification of item sets that are frequently bought together is tried. A FIM algorithm scans the database, possibly multiple times, and finds item-sets that occur in transactions more frequently than a given threshold. The frequency of items that present in a transaction is called *support.*

In this approach , we study whether we can adapt the existing CPU-based FIM algorithms to new-generation graphics processing units (GPUs). 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 scalability and portability, since programs are oblivious about physical cores and rely on hardware for thread management. The *Apriori*algorithm[4] is not only applied in frequent itemset mining or association mining, but also in other data mining tasks, such as clustering , and functional dependency . FIM algorithms are optimized for the location of the data in database. These characteristics may hurt the efficiency on the GPU since SIMD operations favor aligned and sequential data accesses . Apriori or EClatimplementationon the GPU[1],[2],[3]is a quite challengeable, a bitmap data structure is used to represent transactions in GPU-based FIM implementations. Specifically, the bitmap stores the occurrences of items in transactions, and is efficient to be partitioned to SIMD processors. Furthermore, here is utilization of a lookup table to facilitate support counting, which is usually the most time consuming component in the *FIM* algorithm.

Let A be a set, called setof itemsor alphabet. Any subset$X \in P(A)$ofAiscalled an itemset. Let $T \subseteq P(A)$ be a multiset of item-sets, called transaction database, and its elements $T \in T$ called transactions. For a given itemset $X \in P(A)$, the set of transactions that contain X

$$T(X) := \{T \in T \mid X \subseteq T\}$$

is called (transaction) cover of X in T and its cardinality

$$\sup(X) := |T(X)|$$

T

(absolute) support of X inT. An(all) frequent item-set mining task is specified by a datasetTand a lowerbound $\text{minsup} \in N$ on support, called minimum support, and asks for enumerating all itemsets with support at least

minsup, called frequent or (frequent) patterns.

A tremendous growth of data that needs to be processedin 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. Association rule learning, classification, clustering, and regression commonly need to mine data. Let us fix notations for the frequent itemset mining problem in the rest of this section.

A. General Purpose GPU Computing

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 , databases[7] , 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. The current GPGPU technologies include OpenCL[5,6] and CUDA[1,2,3] (Compute Unified Device Architecture).

GPU is a high-performance computing device which does not only reduce the deployment cost but also saves on maintenance. GPU programming strategies can be classified according to either graphic APIs or GPU programming language. It is difficult for developers to use the graphic APIs since they need understand the graphic hardware and encode their data to graphic vectors. NVIDIA and ATI have been proposed as GPU programming language by CUDA and Stream respectively. CUDA can only be used on NVIDIA's GPU. Therefore, OpenCL was proposed in 2009 to deal with platform heterogenity. The program design with OpenCL not only can be executed on different brand GPU device but also on multi-core CPUs.

While GPGPU programming frameworks greatly reduce the complexity of GPGPU computing, developers must carefully design and implement their algorithms in order to fully utilize the GPU architectural features. Furthermore, as a co-processor, the GPU relies on the CPU for memory allocation.

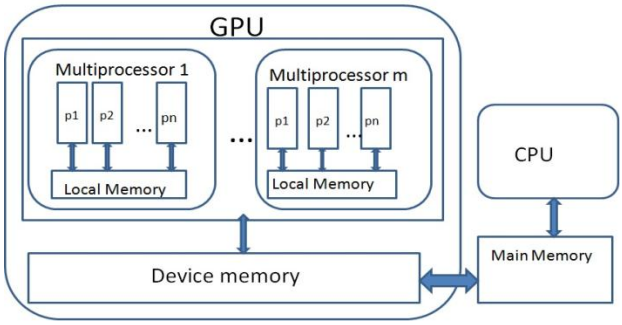


Fig. 1. The many-core architecture model of the GPU

B.OpenCL:

The GPU programming language[5,6] can be classified as graphic APIs (DirectX, OpenGL, etc.), GPU programming language (NVIDIA CUDA [9], ATI Stream [10], OpenCL [11], etc). Previously, GPU programming required developers with in-depth knowledge of graphics programming and hardware. In order to utilize the computation resources on GPU, developers
had to encode data to a graphic vector, and then use the DirectX or OpenGL functions to perform rendering. After that, the rendered data had to be decoded. This procedure not only required graphic programming knowledge, but also depended on different GPUs. Recently, CUDA and Stream have been proposed by NVIDIA and ATI. Both of them provide C interface and allow developers to adapt the hardware, e.g., number of processing units, size of local and global memory. However, previous frameworks could only be used with the respective GPUs, e.g., CUDA could only be executed on NVIDIA's GPUs.

In order to solve this situation, the Khronnos Group and many industry-leading companies created the OpenCL. OpenCL is an open and cross-platform parallel heterogeneous
programming system. It provides a uniform programming environment for developers to write efficient and portable codes using a diverse mix of multi-core CPUs, GPUs, and other processors.

The preliminary work of OpenCL(Open Computing Language) was finished by AMD, IBM, Intel, and NVIDIA while it was initially developed by Apple Inc. OpenCL is a framework which allows C program development in heterogeneous platforms; that is, the framework can be applied in any system which is composed of different CPUs, GPUs, and other computing platforms. It is able to perform in different operating systems as long as the OpenCL library is installed. The CPU and GPU can then communicate with each other and work together by applying the appropriate C++ file for CPU and Kernel file for OpenCL on GPUs to perform parallel computation with GPU.

C.Representing the data in Layouts

There are two layouts that algorithms usually employ to represent transaction databases: 1. horizontal layout 2. vertical layout.In the vertical layout, each item$i_j$in the item base B is represented asi_j: $\{i_j, t(i_j)\}$and the initial transaction database consists of all items in the item base.

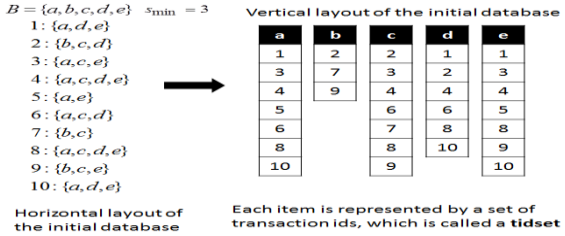


Fig. 2. Layouts for illustration of the data

D.Eclat Algorithm

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 itemsetAcan be easily calculated by cover's intersectionof 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.Candidate Generation and support counting for frequent itemset mining using Eclat algorithm are entirely performed on GPU so as to increase the speed of process and enhance the performance.

Most frequent itemset mining algorithms as Apriori and Eclatuse a total order on the items A of the alphabet and the itemsets P(A) to prevent that the same itemset, called candidate, is checked twice for frequency. Items or-derings ≤ are in one-to-one-correspondence with item codings, i.e., bijective mapso:A→ {1, . . . , n}via naturalordering on N. – For itemsets X, Y ∈ P(A) one defines their prefix as

prefix (X, Y) :={{x ∈ X | x ≤ z} | maximal z ∈ X ∩ Y :

{x ∈ X | x ≤ z} = {y ∈ Y | y ≤ z}}

Any order on A uniquely determines a total order on P(A), called lexicographic order, by

X < Y :⇔min(X\prefix (X, Y)) < min(Y \prefix (X, Y))

For an itemset X ∈ P(A) an itemset Y ∈ P(A) with

X ⊂ Y and X < Y is called an extension of X. An extension Yof X with Y = X ∪ {y} (and thus y > max X) is called an 1-item-extension of X. The extension rela-tion organizes all itemsets in a tree, called extension tree or search tree.

Eclat starts with the empty prefix and the item-transaction incidence matrix $C_\emptyset$, shortly called incidencematrixin the following, and stored sparsely as list of itemcovers: $C_\emptyset$:= {(x, T ({x})) | x ∈ A}. The incidence matrix is filtered to only contain frequent items by

freq(C) := {(x, T_x) | (x, T_x) ∈ C, |T_x| ≥ minsup}.

that represent frequent 1-item-extensions of the prefix. For any prefix p ∈ P(A) and incidence matrix C of frequent 1-item-extensions of p one can compute the incidence matrix C_x of 1-item-extensions of p ∪ {x} by intersection rows:

C_x := {(y, T_x ∩ T_y) | (y, T_y) ∈ C, y > x}

where (x, T_x) ∈ C is the row representing p ∪ {x}. C_x has to be filtered to get all frequent 1-item-extensions of p ∪ {x} and then this procedure is recursively iterated until the resulting incidence matrix C_x is empty, signaling that there are no further frequent 1-item-extensions of the prefix.

The algorithm is shown below:

Input: D, K, i ⊆I

Output: F[I](D, K)

1: F[I] :={}

2: for all iε I occurring in D do

3: F[I] := F[I] ∪{I ∪{i}}

4: // Create Di

5: Di: = {}

6: for all jε I occurring in D such that j>I do

7: C := cover({i}) ∩ cover({j})

8: if |C| >= K then

9: Di: = Di ∪ {(j, C)}

10: end if

11: end for
12: //Depth-first recursion
13: Compute F[I ∪ {i}](Di, K)
14: F[I] := F[I] ∪ F[I ∪ {i}]
15: end for

In this algorithm each frequent item is added in the output set. After that, for every such frequent item i, the i-projected database Di is created. This is done by first finding every item j that frequently occurs together with i. The support of this set {i, j} is computed by intersecting the covers of both items. If {i, j} is frequent, then j is inserted into Di together with its cover. Thereordering is performed at every recursion step of the algorithm between line 10 and line 11. Then the algorithm is called recursively to find all frequent itemsets in the new database Di.

II.Literature Survey

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. FerencBodon implemented Apriori using trie-based data structure & candidate hashing[10], Christian Borgelt implemented Apriori in his work[16] using recursion pruning. The BorgeltGelat 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 implementations:

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 platformis 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 speedup relative to sequential Eclat implementation executed on a multicore CPU.

III. Implementation Details

A.ProjectedAlgorithm :GPU-Eclat

Input: A database D and a minimum threshold value.
Output: a complete set of frequent itemsets .

---

1. Start the CL program to be executed by the GPU.
2. Load D from disk.
3. Generate vertical Tidlist via scanning *D* and store it on CPU memory.
4. Allocate memory space in GPU for Tidlist
5. Stores array Tidlist into GPU memory
6. Sort Tidlists according to bits position
7. Split Tidlist according to sorted bitmap
8. Generate trie containing the *k*-suffix of all the bitsets starting with that bit
9. Join all tries to generate candidate set
10. Store candidate patterns in GPU
11. Allocate memory space in GPU to save the results
12. Perform launch kernel of CL Prog (on GPU)
    a. Each processing unit (PU) of GPU allocated a set of candidate itemsets (CIs)
    b. for each CI in CIs
    i. PU compute the support of CI .
    ii. If support of Candidate item set is greater than or equal to given threshold , then set "it is frequent" , else "it is not frequent".
13. Wait until GPU finishes its program execution.
14.Retrieve the results from GPU and save them in CPU memory
15. Repeat the process from step 8 to 14 until all the same level candidate patterns are done.
16. Move to next level candidate set generation and perform Steps 8-15 until all candidates are generated and verified.

Candidate Generation:

Input : Database D
Output: Candidate itemsets

Step1 :Scan the database D and convert it into vertical transaction list.
Step2: for all i□I
Step 3: Di: = {}
Step 4: for all j □I occurring in D such that j>I do
Step 5: C := cover({i}) ∩ cover({j})

Freq-Itemset Generation (Support Counting)

Input: Candidate trie, min. threshold
Output: Frequent Itemset

1://u represents a node at depth K ¡ 1 in the trie.
2: for each u at depth K ¡ 1 do
3: for each w that is a right sibling of u do
4: //Join
5: Union on the two (K - 1)-itemsets represented by
u and w to obtain a candidate K-itemeset
6: //Pruning

7: (K - 1)-subset test on the candidate K-itemset by
following the path of the trie with the same prefix
8: end for
9: end for

B.Steps for implementation of current approach:

1.Data preprocessing :Before initialization data preprocessing is necessary to execute.It takes 3steps to preprocess the dataset for the purpose of reducing memory usage.Transactions , stored in horizontal format , are read from disk and converted to vertical format using the (item, bitvector) representation. Because infrequent items will not appear in any frequent itemset, they can safelybe removed from the dataset without altering the FIM results. In the second step,the frequency of each item is counted and the infrequent items and their

correspondingvertical lists are deleted from the vertical list.The remaining items will be sorted by frequency from low to high and remappedto build a better balanced expansion search space.Then the candidate generation step is executed by using the mentioned algorithm and then support counting is done with selecting frequent itemsets .

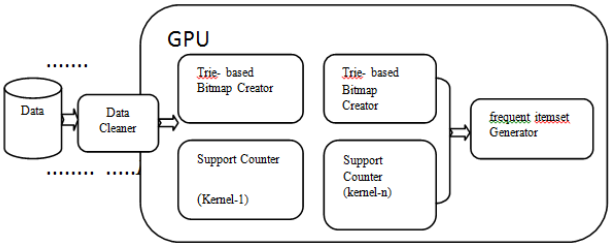
C.Project Design

Experiments are conducted to verify the performance of the approach on GPU. The languages used are OpencL and C++ on visual studio while the operating system is Microsoft Windows 7.

Table1.Hardware and Software Configurations:

Item	Description
CPU	AMD A8-4500M 1.9GHz
Memory	4GB memory
GPU	RADEON DUAL GRAPHICS HD 7640G+7470M and 1GB memory
OS	Microsoft Windows 7
Compiler	Microsoft Visual Studio C++ 2008
SDK	OpenCL 1.1

***D.*SystemArchitecture :**



....Fig.3. System architecture

System architecture contains Database for storage ,data Cleaner for data preprocessing , Trie based bitmap creator and Support counter on GPU and Frequent Itemset generator which will find the frequent itemsets by using Trie based bitmap creator and Support counter result aggregation.

IV. Results:

A.Comparative Analysis:

Data set: The dataset is obtained from the UCI repository of machine learning databases . The characteristics of adult dataset selected for comparison of apriori and Eclat algorithm:

Table 2. Characteristics of adult dataset

File name	No.of records	No.of Columns
Adult.D14.N48842.C2.num	48842	14

Table 3.Difference between Apriori and Eclat algorithm

Parameters	Apriori Algorithm	Eclat Algorithm
Technique	It uses a breadth-first search approach and uses apriori property (All nonempty subsets of a frequent itemsets must be frequent) and join–prune method	It uses a depth-first search approach and uses intersection of transaction ids list for generating candidate itemsets
Memory Utilization	Due to large amount of candidate are produced so require large memory space	Require less amount of memory compare to apriori if itemsets are small in number
Databases	Suitable for sparse datasets as well as dense dataset.	Suitable for medium and dense datasets but not suitable for small datasets.
Time	Execution time is more as time wasted in producing candidates at every time.	Execution time is small than apriori algorithm.

Table 4. gives the total execution time of Apriori, Eclat algorithms with different support threshold using an adult data set.The execution time is decreased when the support threshold increased. Experiment is done with minimum support ranging from 30% to 70%.The Eclat algorithm outperforms Apriori algorithm.

Table 4. Total Execution time using Adult dataset

Support	Total Execution Time in Seconds	
	Apriori	Eclat
30	9.85	0.54
40	6.72	0.49
50	4.51	0.45
60	2.69	0.44
70	1.7	0.4

Table 5. Three experimental datasets

Dataset	#Item	#Transactions	Characteristics	Data size
T40I10D100K	1,000	100,000	Synthetic	~15 MB
Retail	16,469	88,162	Sparse/Real	~4 MB
Chess	75	3,196	Dense/Real	~335 KB

1.PBI Pure Bitmap Based Implementation: Processing entirely on GPU
→Candidate Generation & Support Counting on GPU
→Itemsets&Transactions: Bitmap data Structure

2.TBI Trie Based Implementation: CPU/GPU Coprocessing
→Candidate Generation on CPU & Support Counting on GPU
→Itemsets: Trie& Transactions: Bitmap Data Structure

3.GPU Eclat (Frontier Expansion) Implementation
→Candidate Generation on CPU Using Stack & Support Counting on GPU
→Bitset Representation for vertical transaction list

Table 6.Comparison of PBI-GPU AND TBI-GPU

Sr. No.	**TBI-GPU Based Approach**	**PBI-GPU Based Approach**
1.	Candidate Generation: single threaded on CPU	Candidate Generation: Multi threaded on GPU
2.	Support Counting: Multi threaded on GPU	Support Counting: Multi threaded on GPU
3.	Representation: Itemsets: Trie (horizontal data layout) Memory Requirement is low but difficult to parallelize Transactions: Bitmap(helps the GPU SIMD parallelism & Boosts the Performance)	Representation: Itemsets: Bitmap(vertical data layout) Memory Requirement is high but suitable to parallelize Transactions:Bitmap(helps the GPU SIMD parallelism & Boosts the Performance)
4.	TBI-GPU outperforms PBI-GPU on the sparse dataset	PBI-GPU outperforms TBI-GPU on the dense dataset ,due to smaller size of bitmap representation for itemsets

B.ProposedEclat Result:

We evaluatd the visualization performance of Eclat with three different support thresholds 1%, 1.5% and 2%, on the T40I10D100K transaction set.

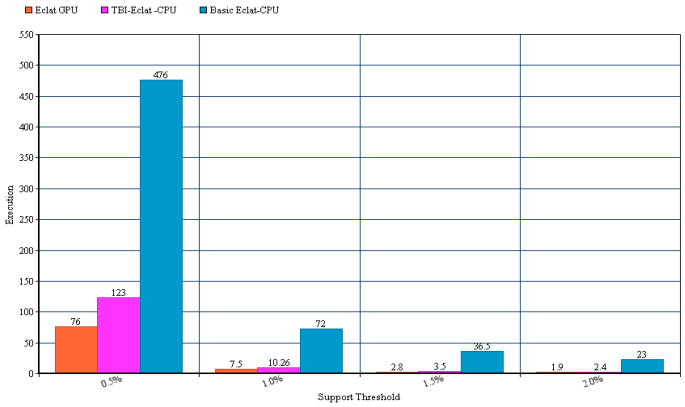


Fig.4.Execution Time (sec) of three implementation of Eclat on datasetT10I4D100K

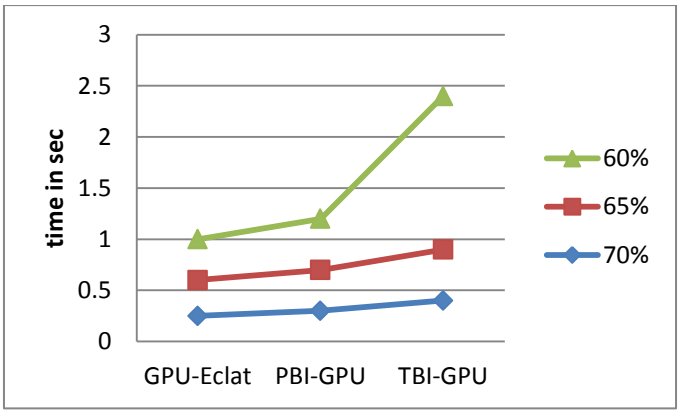


Fig.5. experiments on the dense dataset Chess

VI. Conclusion And Future Work

The GPU-based implementations of *ECclat*algorithm for frequent itemset mining has been presented . The implementation employ a bitmap data structure to encode the transaction database on the GPU and utilize the GPU's SIMD parallelism for support counting. The implementation stores the itemsets in a bitmap, and runs entirely on the GPU. i.e. vertical transaction lists are represented using "bitset representation ,operated using wide bitwise operations across multiple threads on a GPU. The previous evaluation results [1,2,3]show that both of GPU- implementations are up to two orders of magnitude faster than optimized CPU-based implementations , and compared with our current GPU Eclat approach which is a faster method than TBI- GPU,PBI-GPU , TBI Eclat CPU and Eclat CPU.

It is a challenging to investigate the data compression techniques & development of a buffering mechanism between the GPU memory and the CPU memory . Future work will be to explore other mining algorithms with GPU acceleration, for instance, FP-growth and classification.

References

[1] WenbinFang ,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. BakosGPApriori : 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 ,WenOuyang, Jiayi Zhou An OpenCL Candidate Slicing Frequent Pattern Mining Algorithm on Graphic Processing Units[2011 IEEE National Science Council]

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

[8] RakeshAgrawal and RamakrishnanSrikant.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] FerencBodon. A fast apriori implementation.*FIMI*, 2003.

[11] ShuaiChe, Michael Boyer, JiayuanMeng, David Tarjan, Jeremy W. Shea®er, and Kevin Skadron.A performance study of general-purpose applications on graphics processors using cuda.*Journal of parallel andDistributed Computing*, 2008.
[12] http://fimi.cs.helsinki.fi/. *FIMI repository*.
[13] http://www.adrem.ua.ac.be/goethals/software/files/apriori.tgz. *Aprioriimplementation from Bart Goethals*.]
[14] ShubhabrataSengupta, Mark Harris, Yao Zhang, and John D. Owens. Scan primitives for gpu computing. In *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 [2008] A new parallel algorithm for the frequent itemset mining problem In:International Symposium on parallel & distributed computing, 2008, ISPDC '08, PP 165-170
[18] Ansari G, DastghaiGifardG(2008) Distributed FrequentItemset mining using trir data structure. Int J Computer Sci.35(3): 377-381

Sarika S. Kadamgraduated in Computer Engineering from the V.T.U. University of Belgaon (India) in 2008. She is pursuing her Master of engineering in Computer Engineering from the University of Pune (India).She is currently research scholar student in Computer Department, PimpriChichwad College of Engineering; Pune (India).Her research interests include wireless networking, Parallel computing and Distributed System.

SudarshanS. Deshmukh graduated in Computer Engineering from the University of Shivaji (India) in 2004. He received his Masters in Computer Engineering from the Bharati University in 2009. He is currently working as an assistant professor, Computer Engg, at PCCOE, University of Pune since 2009. He is a member of the Technical Committee of Parallel Processing (TCPP), IEEE communication society, IAENG etc. Received Nomination for IEEE Technical Committee on Parallel Processing Outstanding Service Award for 2011.Associate Editor of International Journal of Cloud Applications and Computing, also serving as reviewer to several journals and conferences His research interests include distributed systems, resource sharing, load balancingetc.