SCALABLE FREQUENT ITEMSET MINING USING HETEROGENEOUS COMPUTING: PARAPRIORI ALGORITHM

V. B. Nikam and B. B. Meshram

Department of Computer Engineering
Veermata Jijabai Technological Institute, Matunga, Mumbai, India

ABSTRACT

Association Rule mining is one of the dominant tasks of data mining, which concerns in finding frequent itemsets in large volumes of data in order to produce summarized models of mined rules. These models are extended to generate association rules in various applications such as e-commerce, bio-informatics, associations between image contents and non image features, analysis of effectiveness of sales and retail industry, etc. In the vast increasing databases, the major challenge is the frequent itemsets mining in a very short period of time. In the case of increasing data, the time taken to process the data should be almost constant. Since high performance computing has many processors, and many cores, consistent run-time performance for such very large databases on association rules mining is achieved. We, therefore, must rely on high performance parallel and/or distributed computing. In literature survey, we have studied the sequential Apriori algorithms and identified the fundamental problems in sequential environment and parallel environment. In our proposed ParApriori, we have proposed parallel algorithm for GPGPU, and we have also done the results analysis of our GPU parallel algorithm. We find that proposed algorithm improved the computing time, consistency in performance over the increasing load. The empirical analysis of the algorithm also shows that efficiency and scalability is verified over the series of datasets experimented on many core GPU platform.

KEYWORDS

Association Rules Mining, Frequent Itemsets Mining, Scalability, Parallel Data Mining, GPU Computing

1. INTRODUCTION

The exponential growth of technology used in social, business, and scientific domain, database sizes has made difficult to interpret meanings out of generated data. In this process, data mining plays vital role in automatically extracting useful and hidden information from such large databases [22]. Agrawal et. al.[12] first presented association rules mining for finding frequent itemsets on market basket analysis. The performance for generating frequent itemsets will subsequently decrease, or to compensate with the performance, candidate set need to reduce by making use of sampling, pruning techniques, which may affect to accuracy of the results due to reduction in the training data sets.

The promise of Data Mining is that it delivers technology that will enable the development of new breed of decision support application; however the delivery performance will be a bottleneck unless we prefer parallel computing. The major factors will influence the parallel mining of Association Rules are; Availability of very large data sets, Memory limitations of sequential computers, Since databases to be mined are often measured in gigabytes and even in terabytes, parallel algorithms [14] would be required. For data intensive applications, rather than going for traditional shared nothing architecture; Cloud, General Purpose computation on Graphics Processing Units, the inexpensive that giving largest computing power per dollar and highly scalable technologies are widely accepted today. Developing algorithms for GPU computation is challenging, since the architecture imposes many requirements on the way algorithms work, if the potential is to be fully utilized. The availability of hundreds of processing units has resulted in large speed up and high scale-up [2]. It is well known that large speed gains can be obtained by using a graphics processing unit (GPU) as a massively parallel computing device.

This paper is organized as follows; section2 discusses the detailed literature on Association rule mining and GPU architecture. The proposed parallel and scalable Aprioiri algorithm approach on General Purpose Graphics Processing Units is discussed in section3, the experimental work on Graphics Processing Units (GPU) the massively parallel processing engines and result analysis are discussed in section4. In section5 we concluded our work of implementing the scalability aspects of parApriori algorithm.

2. LITERATURE SURVEY

In the association rule mining algorithms, the analysis is to be done on the candidates, which are computationally very large. To accommodate the candidates for the computations, sampling is preferred so far, but it may affect on accuracy. There are algorithms suggested without generating candidate set also, but still has to compromise with the computing efficiency. Sharing the many processor's compute capability could be the efficient solution. Considering the many issues, it is found that Apriori seems to be more convenient for parallelism.

Key Definitions:

Itemset: A collection of one or more items *i* used in transactions *T* of a database *D*, e.g. {Milk, Bread, Diaper}, An item set that contains '*k*' items, called as k-itemset.

$T = \{ i_1, i_1, i_1, i_{1\ldots} \}, T \in D.$

Frequent Itemsets: The sets of item which occurs above minimum predefined cut-off. This minimum cut-off is called as minimum support threshold 'σ'. Frequent item set is denoted by L_i for i^{th} -Itemset.

Frequent Itemset : support of itemsets is equal or higher than predefined *minsup* threshold.
Candidate Itemsets: The itemsets, C_k for k^{th} itemset, which are required to find out frequent itemsets, called candidate itemsets. The candidate itemsets includes all the items of the transactions. This can be generated using L_{k-1} joining with itself.

Support count (σ): Frequency of occurrence of an itemset defined by the user to find frequent itemsets. e.g. σ ({Milk, Bread, Diaper}) = 2.

Support: Fraction of transactions that contain an itemset, e.g. s ({Milk, Bread, Diaper}) = 2/5.

2.1. Sequential Apriori Algorithm

In association rules mining, we focus for the Apriori algorithm. Our proposed parallel algorithm is built on top of the sequential Apriori, association rules mining algorithm [20]. The generic Pseudo code of the Apriori algorithm is discussed in Figure1.

Figure 1.Pseudo Code for Apriori Algorithm

Algorithm : Apriori Algorithm
Input: Transaction Database, D; Item T, Minimum Support S_{min}
Output: Frequent Item sets, F
Cj: candidate item sets of size j, Fj: frequent item sets of size j.

```
Apriori (D, T, Smin)                          // Apriori algorithm
    begin
        k := 1;                               // initialize the item set size
        $C_k := \bigcup_{i \in D} \{\{i\}\}$;            // start with single Candidate
        $F_k$ := prune ($E_k$, T, Smin)              // determine whether item is frequent?
        While $F_k \neq \emptyset$ do begin              // repeat till the frequent item sets available
            $C_{k+1}$ := candidate ($F_k$ )          //create candidate with 1 item more
            $F_{k+1}$ := prune ($C_{k+1}$, T, Smin)    // determine whether frequent item sets
            k := k+1;                         // increment the item counter
        End
        Return $\bigcup_{j-1}^{k} F_j$                       // return the frequent item sets
    End
```

During iteration *k* of the algorithm, a set of candidate *k*-itemsets is generated. The database is then scanned and the *support* for each candidate is found. Only the frequent *k*-itemsets are retained for future iterations, and are used to generate a candidate set for the next iteration. A pruning step eliminates all candidate sets which has an infrequent subset. This iterative process is repeated, until there are no more frequent *k*-itemsets to be found. In Figure 1, L_k denotes the set of frequent *k*-itemsets, and C_k the set of candidate *k*-itemsets. There are two main steps in this algorithm,

1. *Candidate itemset* generation and
2. *Support* counting.

2.1.1. Candidate Itemset Generation

In candidate itemsets generation, the candidates C_k for the *k*-th pass are generated by joining L_{k-1} with itself, which can be expressed as

$$C_k = \{ x | x[1:k-2] = A[1:k-2] = B[1:k-2],\ x[k-1] = A[k-1], x[k] = B[k-1], \\ A[k-1] < B[k-1], \text{ where } A,B \in L_{k-1} \}$$

Where $x[a:b]$ denotes items at index *a* through *b* in itemset *x*. Before inserting *x* into C_k we test whether all $(k-1)$-subsets of *x* are frequent. If there is at least 'one' subset that is not frequent the candidate can be pruned. Collecting the frequent item sets of size *k* in set L_k has drawbacks, i.e. a frequent item set of size *k+1* can be formed in $j = k(k + 1)/2$ possible ways. A core problem is that an item set of size '*k*' can be generated in '*k*!' different ways. As a consequence, the candidate generation step may carry out a lot of redundant work, since it suffices to generate each candidate item set once. Can we reduce or even eliminate this redundant work? and this is the computational challenge !

Support Counting: To count the support of candidate *k*-itemsets, for each transaction *T* in the database, we conceptually form all *k*-subsets of *T* in lexicographical order. For each subset we then traverse the ordered dataset and look for a matching candidate, and update its count.

$$\text{Support (item } i) = \frac{\text{\# tuples containing item 'i'}}{\text{Total number of tuples in a transactional database 'D'}} .$$

It means *support, s,* is the probability that a transaction contains {*i*}.

2.1.2. Rule generation

The second phase is to generate rules from frequent item sets. Generating rules is much less expensive than discovering frequent itemsets, as it does not require examination of data.
Given the frequent keyword set *l,* rule generation examines each non empty subset '*a*', and generates rule

$$r \Rightarrow (l\text{-}a) \text{ on support}(l), \text{ for } \quad \text{Confidence} = \frac{\text{support (l)}}{\text{support (a)}}$$

2.2. Related Work in Sequential Environment

In general, much of the work in frequent itemset mining algorithm development is focused on serial algorithms. Three of the best-known frequent itemset mining algorithms are Apriori [5],[6], Eclat [7] and FP-Growth [8]. Apriori, 1. Iteratively generate *k*+1-sized *candidate generation*, 2. After generating each new set of candidates, the algorithm scans the transaction database to count the number of occurrences of each candidate. This step is called *support counting.*

The primary difference between Apriori and Eclat is the way they represent candidate and transaction data; and the order that they scan the tree structure that stores the candidates. FP-Growth is the most recently-developed algorithm; the main difference from the previous two approaches is that FP-Growth doesn't generate candidate sets iteratively. Though FP-Growth is faster than Apriori and Eclat, for high support threshold, Apriori outperforms FP-Growth[9]. Among the few out performed implementations, Ferec Bodon implemented Apriori using trie-based data structure and candidate hashing [10], Christian Borgelt implemented Apriori in his work [11] using recursion pruning, Bart Goethals implemented Apriori based on Agrawal's algorithm [6] Comparison between those implementations can be found in Bodon's work[15]. Bodon presented[19] faster implementations for frequent itemsets mining problem. These implementations are the racing for the performance in sequential algorithm, and preferably chosen Apriori algorithm for the research on frequent itemset mining.

2.3. Fundamental Problems in Apriori algorithm

Apriori is processed for the tasks of, frequent itemset generation and association rules findings. The major and performance crunching task is frequent itemset generation, because of the large itemset generations. Frequent itemset generation is the performance bottleneck. The Apriori algorithm uses frequent (k – 1)-itemsets to generate candidate frequent k-itemsets. This uses repeated database scan and pattern matching for support findings in the candidate itemsets. For the 10^4 items datasets, in the worst scenario there could be 10^4 frequent 1-itemset will be generated, which further may generate 10^7 candidate 2-itemsets, and so on till there are no frequent itemsets remained from the candidate set. It needs to generate $2^{100} \approx 10^{30}$ candidates for 100, viz $\{a_1, a_2, \ldots, a_{100}\}$ item size frequent pattern. In addition, this candidate generation takes multiple scans; for '*n*' length datasets, it takes (n +*1*) scans.

Apriori has been observed for the inefficient and more I/O, also produces much more useless candidates. To improve Apriori's efficiency, the methodologies already discussed are, *Transaction reduction, Partitioning, Sampling, Dynamic itemset counting, Hash-based itemset counting, Mine the rules without Generating Candidate* sets, *Increase* of minimum support.
These methods can be implemented by reducing database size at each iteration, prune useless candidates in advance, making use of efficient data structure, generate frequent itemsets without candidate generations, etc. Using these means the Apriori's efficiency is improved using Eclat,

Frequent-Pattern Growth, COFI tree(faster FP-Growth). However, these are improvements are focused to sequential developments of the algorithms to improve the efficiency.

2.4. Related work in Parallel Environment

Bodon's implementation for parallel computing is revised by Yanbin, et. Al [12] keeping the data storage methodology and algorithm same. The performance is achieved by implementing a disjoint partitioning of datasets on *symmetric multiprocessing* computer. They proved partitioning a transaction database and fitting each partition into limited main memory for quick access and allowing incremental generation of frequent itemsets improves the performance of frequent itemsets mining. Puttegowda D, et. Al[13] used shared-nothing architecture where MPI are used as the communication primitives. Data is evenly, *1/N* partition, distributed on the disks attached to the processors, they used data level parallelism. Each processor P_i receives a *1/N* part of the database, processor P_i performs a pass over data partition D_i, develops local support count for candidates in C_k. Each processor P_i now computes L_k from C_k, and makes the decision to terminate or continue to next pass. Parthasarathy et al[15] addressed the Hash Tree structure for data storage and load balancing. They have parallelized the Hash tree on shared memory architecture for association rule mining. Rasmus Resen et al[16] chosen GPU for achieving parallelism and addressed the data partitioning layout for each thread to compute with. They have proved suggested data layout as a best partitioning for set intersection problems. George Teodoro el al,[17] proposed tree projection-based frequent itemset mining algorithm. They further extended for parallelization opportunities of the algorithm to shared-memory multi-core, and the GPU environments. Distributed Data Mining (DDM)[3][4][21] works on partitioning the data, applying the data mining algorithms to each local subset on processor and then combining the local knowledge discovered by the algorithms into a global knowledge. But such global knowledge is usually less accurate as the number of subsets increases [19].

2.5. GPGPU as Parallel Platform

Recent developments of GPGPU have made low cost high performance and scalable computing for any applications. In addition, GPGPU supported programming models better exploit the parallel power of the many core GPU architectures. Present Data mining tools are not able to meet the requirement of large-scale databases in terms of speed and scalability. GPU technology is the extensive scalable platform to deploy the data mining algorithms.

Due to the inherent parallelism of GPU many core architecture, it has become possible to program GPU processors directly, as massively parallel processors. GPU computation is highly scalable, inexpensive and high performance per dollar. Combining CPU with the GPU massively parallelism as abstracted in 0 helps to scale up the algorithms for knowledge discovery by applying the data mining algorithm on the entire dataset.

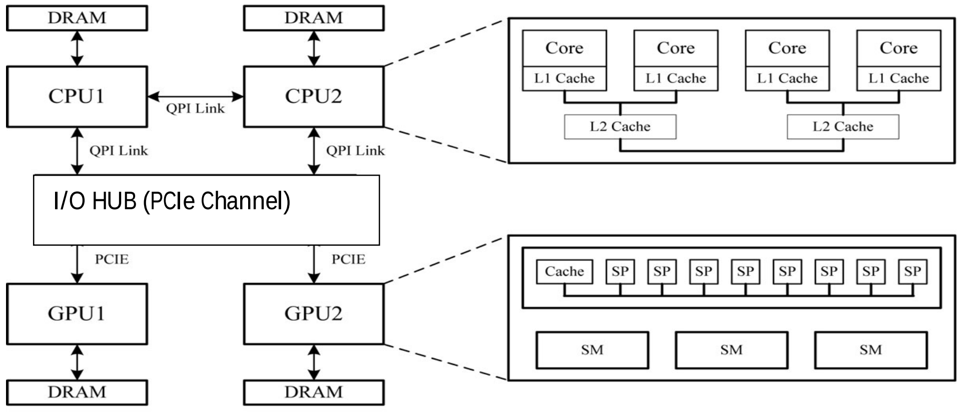


Figure 2.CPU+GPU Scalable and Parallel Architecture

The GPU architectfure has Streaming Multiprocessors, which has many blocks. Each block has multiple threads. ˄ll these threads executes on concurrent parallelism. We target for the GPU implementation for Apriori Algorithm to implement the scalability of the algorithm. 0, explores the CUDA memory hierarchy, which helps to exploit the parallelism on large datasets. The data access on device memory, global memory, texture memory, and constant memory varies the performance of the algorithm. Also, CUDA environment gives flexibility to share combined memory, offers very less communication overheads.

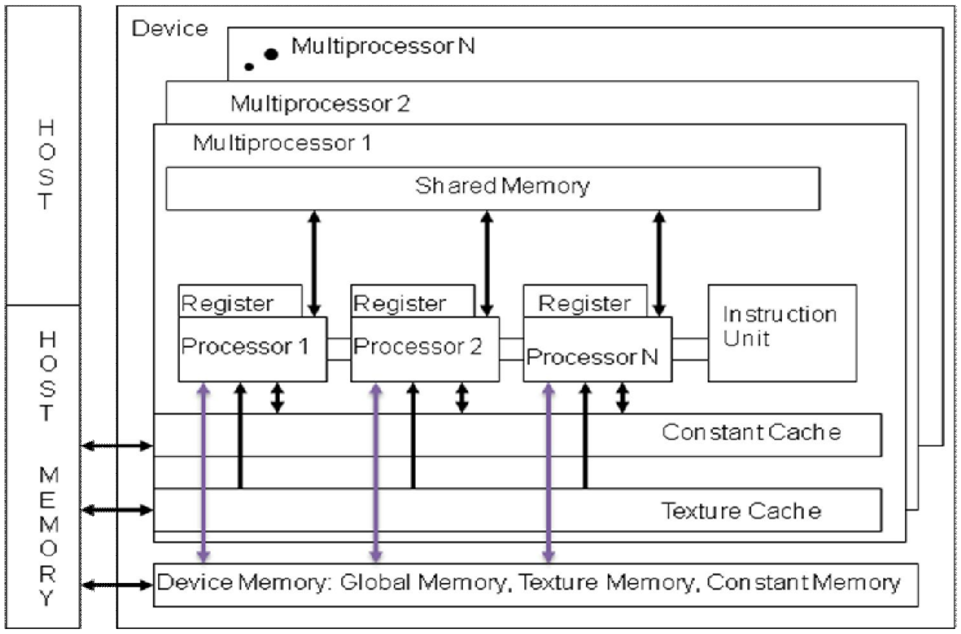


Figure 3.NVIDIA GPU - Memory Hierarchy

The maximum possible threads to be active for parallel execution can be computed by occupancy formula as given below.

$$\text{Occupancy} = \frac{\text{blocks per SM} \times \text{threads per block}}{\text{maximum threads per SM}}$$

Where SM is the acronym used for Stream Multiprocessors in GPU architecture.

CUDA threads are Lightweight and Fast switching, 1000s execute simultaneously. All threads execute the same code, each thread has an ID, Threads are grouped into blocks, Blocks are grouped into a grid and A kernel is executed as a grid of blocks of threads.

3. PROPOSED PARALLEL APRIORI ALGORITHM : ParApriori

Given *m* items, there are potentially 2^m frequent item sets; however, only a small fraction of the whole space of itemsets is frequent. Discovering the frequent item sets requires a lot of computation power, memory and I/O, which can only be provided by parallel environments [15].

3.1. Data Representation for Parallel Apriori Algorithm

Apriori algorithm assumes that candidate sets to be in memory, which can be expensive when the candidate set is very large. Concerning speedup, memory optimized usage and sensitivity of parameters, the data structure to accommodate the complete transactional data, and search efficiency, the researchers [23] has suggested Trie, Hash Tree. However tree structure is not convenient as compared to the transactional data; as transactional dataset is always the best candidate to parallelize the transactions.

Horizontal Representation: The most straight forward way to store transactions is to store a list of items that comprise each transaction, shown in Figure 4 : 0(a). This is called the horizontal representation.

Vertical Representation: Stores the list of the transactions *ids* corresponding to items, shown in Figure 4 : 0(b). The vertical representation has been referred by variants of Apriori algorithms. Experimental results show that the vertical representations usually can speed up the algorithm by one order of magnitude on most of the test dataset. This approach is referred to as a "*Tidset*". This method of candidate generation is called Equivalent-Class clustering [24]. This vertical data structure is *diffset*. Zaki and Gouda[7] first introduced diffset to reduce the memory requirement of the vertical tidset representation. It speeds up candidate generation by avoiding the slow $O(n^2)$ complete join[25].

BitMap Representaion: The transaction list can also be represented as a bit corresponds to the transactions and item *id*, as shown in Figure 4 : 0(c)., which we can refer as a "*bitset*". When the candidates are represented as bitsets, it takes comparatively less memory space than horizontal and vertical representations, as it takes only *Byte* per item for representation. However, this could be best effective if the matrix is dense.

Tid↓	Items ->			
1	A	D	E	
2	A	B		
3	C	D	E	
4	A	B	D	E

Figure 4 : 0(a) Horizontal Representation

Items ->	A	B	C	D	E
	1	2	3	1	1
	2	4		3	3
	4			4	4

Figure 4 : 0(b) Vertical Representation (Tidset representation)

↓Tid \ Items →	A	B	C	D	E
1	1	0	0	1	1
2	1	1	0	0	0
3	0	0	1	1	1
4	1	1	0	1	1

Figure 4 : (c) Bitmap Representation

Figure 4 : Data Representations Techniques used for Apriori Processing

As shown in 0(b), Tidsets are stored as linear ordered arrays, and when traversing them during the support counting operation, the resultant memory access pattern and instruction stream branching behavior is unpredictable and leads to poor performance on the GPU. The *Tidset* representation is compact but join operations on *tid*sets are highly data dependent and difficult to parallelize. On the other hand, the bitset representation, as shown in 0(c), requires comparatively less memory space and also it is more suitable for designing a parallel set join operation, which is better suited for GPU. Joining two bit-represented transaction lists can be performed by a "bitwise and" operation between the two bit vectors. We prefer to use the bitmap representation of the transactional dataset.

3.2. Parallel and Scalable Apriori Algorithm using GPGPU

For associations rule mining algorithms, reduce the number of scans, promote small size dense data sources, etc are the solutions to compensate main memory space requirement during processing. CPU reads the dataset and is taken to CPU memory and then transferred to GPU's global memory. The dataset accommodates in the memory; performs k-item frequent itemset on GPU cores parallel. The memory is reused for every next k+1 item frequent itemset finding, by optimizing memory usage. The proposed model as shown in 0 aims to find the frequent itemset for large transactional dataset on many core GPU architecture.

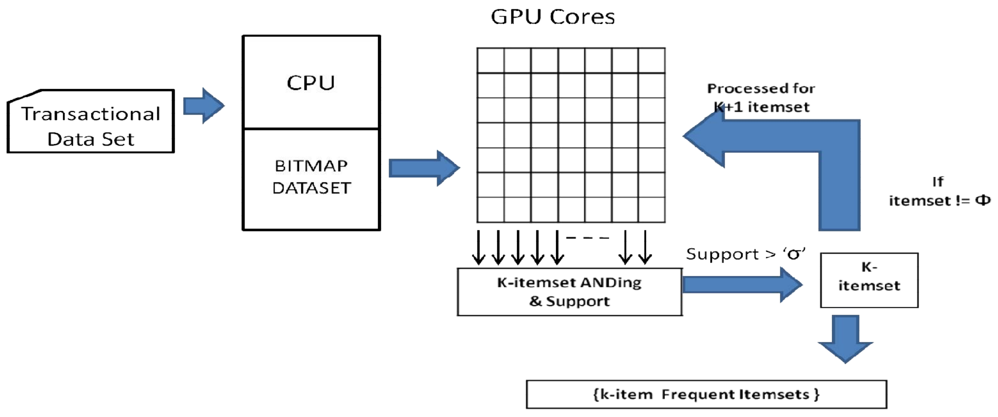


Figure 5. Scalable model for finding Frequent Itemset on GPU cores.

The model finds itemsets generated from the transactional dataset. The dataset accommodates in the memory; performs *k*-item frequent itemset on GPU cores parallel. The memory is reused for every next k+1 item frequent itemset finding, by optimizing memory usage. When the length of frequent pattern is long, the number of maximum frequent itemset increases exponentially which makes the problem computationally difficult. Therefore, here, we have a focus on compute efficiency in algorithm design. The pseudo code of the parallel Apriori algorithm for finding frequent itemsets on GPU is shown in Figure 6:0 below.

Figure 6.Pseudo Code for Parallel Apriori Algorithm

```
Input: Transaction Database, D; Minimum Support- Smin;
Output :Frequent Item sets, FI
        $C_j$: candidate item set, FI: frequent item sets.
        Apriori (D,$S_{min}$)
              Begin
                     While (!Eof ) do begin
                   // Load the Data Set 'D' from Disk,
                          DatasetBuff = ReadFile(row); Row++;
                  // Scan the data set, and find row, columns for the row matrix.
                         FindMaxColmn();
                      End
                    //Allocate GPU memory to accommodate this transaction,
                     Malloc (Matrix_Size)
               // Create a Matrix for 'D' to accommodate the transaction in bit matrix.
                  ConvertToBitMatrix();
                // Allocate memory space in GPU; Transform $S_{min}$ and bitMatrix to GPU
        memory
            End

        // Call GPU Kernel for finding Frequent Itemset
            GPU_Kernel Begin
                     ReadItemSet Cm, Cn
              // ANDing the item transactions and finding support for itemset
                  SupportCount = Sum(Cm &&Cn)
                   If  SupportCount < $S_{min}$
                         prune (Cm,Cn)
                       else
              // Add in frequent item sets list.
                     FI = AddCandidate()
             Kernel End
         Return FI.
```

As shown in Figure:60, each thread processes one row of the bitmap representation. All the threads perform logical AND on the respective item sets and updates the array. Sum is computed for every itemset on all transactional records. If computed sum, is the less than the given support threshold, Pruning is done. The example below, illustrates the working principle of Figure:6 support counting and frequent itemset mining is done on GPU using CUDA environment.

ILLUSTRATIVE EXAMPLE

Let us take a transactional dataset,

Tid	Transactions
1	1,2,4,5
2	1,3,5
3	2,4,5
4	3,4,5

The following is the bit map representation of the transactional dataset shown above.

↓ Tid \Candidate ->	**1**	**2**	**3**	**4**	**5**
1	1	1	0	1	1
2	1	0	1	0	1
3	0	1	0	1	1
4	0	0	1	1	1

Bitmap representation

Finding support for Item sets:

Candidate-> Tid ↓	**1**	**2**	**3**	**4**	**5**	**1, 2**	**1, 3**	**1, 4**	**1, 5**	**2, 3**	**2, 4**	**2, 5**	**3, 4**	**3, 5**	**4, 5**	**1, 5, 2**	**1 , 5 , 4**
1	1	1	0	1	1	1	0	1	1	0	1	1	0	0	1	1	1
2	1	0	1	0	1	0	1	0	1	0	0	0	0	1	0	0	0
3	0	1	0	1	1	0	0	0	0	0	1	1	0	0	1	0	0
4	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0
Support	**2**	**2**	**2**	**3**	**4**	**1**	**1**	**1**	**2**	**0**	**2**	**2**	**1**	**1**	**2**	**1**	**1**

Frequent item set computed with GPU Kernel for *support threshold* (σ) =*2* is
{1,2,3,4,(1,5), (2,4), (2,5), (4,5)}

In this paper, we presented our GPU based method for parallelization of an Apriori algorithm on CUDA platform. Taking advantage of many cores, we proposed three-step technique to speed up Apriori algorithm of data mining on the CUDA platform.

1. *Scalable thread scheduling for logical ANDing scheme for all itemsets*
2. *Parallel distribution many threads for support counting*
3. *Parallel implementation of pruning based on threshold for finding FI*

This method avoids using sampling of data sets to compute frequent itemsets. In case of difficulty in occupation of the dataset in, we may proceed to horizontal partitioning of the dataset and partial support can be computed which can be further merged to compute total support.

4. Experimental Work and Result Analysis

The algorithm assumes the many core GPGPU and CUDA environment for execution of the Apriori algorithm. BARC's supercomputing computing facility and features of NVIDIA GPU Fermi architecture we have used for experiment. Our work is experimented on the ANUPAM-adhya 1U Rack mount server class machine with configuration CPU 6-Core, 2.93GHz, Dual Intel Xeon Processors, 48GB RAM, Two NVIDIA TeslaC2050 Fermi GPUs, and Scientific Linux5.5 GNU with CUDA4.0 toolkit., NVCC compiler & SDK. This work is carried out in at supercomputing division of BARC, Trombay. Experiments are conducted to verify the performance of the frequent pattern finding on heterogeneous platform. The languages used are CUDA-C/C++ on scientific Linux platform. Input data is synthetic dataset generated from IBM data generator. We have used small, medium and large datasets. The datasets we have used are the synthetic datasets. The is statistical characteristic of datasets are shown in Table 1.

Table 1 : Data Sets used for experiments: Database properties.

DataSet	T	I	D	Total size
T5I2D100K	5	2	100,000	19.3MB
T10.I4.D100K	10	4	100,000	4.3MB
T25I20D100K	25	20	100,000	12.1MB
T25I10D10K	25	10	10,000	1.1MB
T40I10D10K	40	10	100,000	5.08MB
T10I4D1M	10	4	100,000	4.1MB
T40I10D100K	40	10	100,000	15.5MB

Statistical Characteristic of Datasets: T- Avg Trans Len, I- Avg Len of Max Patterns, D-No of Trans.

Results Graphs

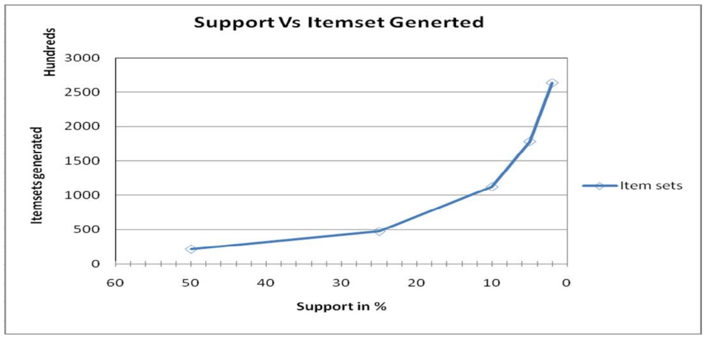
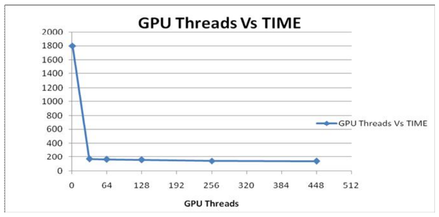


Figure 7. Item sets generated for DataSet=T10I4D1M for various support.



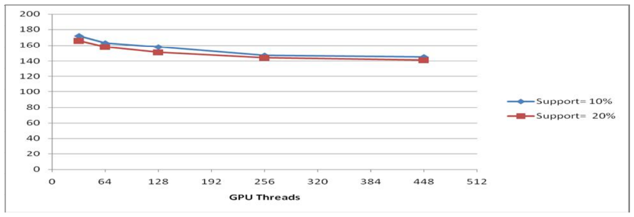


Figure 8. Time is nearly constant for increased load (FIM on supp=10%, 20%)

Figure 9: Almost Proportionate time taken 0by varying load on same compute threads

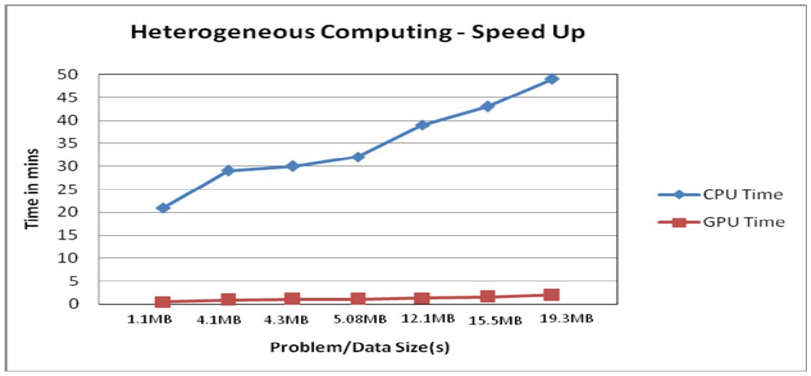


Figure 10: CPU vs GPU time

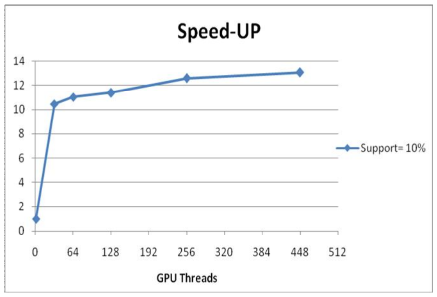


Figure 11: Xfactor speed up

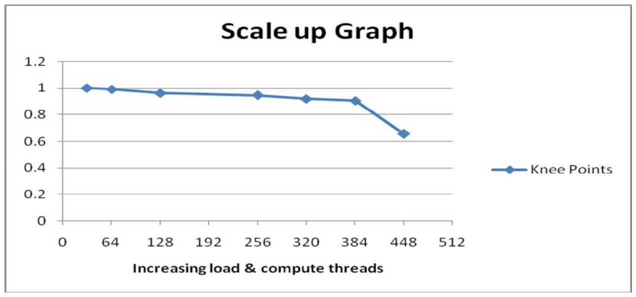


Figure 12: Moving of knee point

Figure7 shows that, as the support value (σ) decreases, the data size of an itemsets increases. We have taken increasing load in terms of increased itemsets by taking small support value, i.e $\sigma \rightarrow 0$. Figure8 shows the drastic reduction in time when thread increased from 1 to 32; however increase of threads from 32 to 448 does not observed reduction in recognizable compute time, it is almost constant and linear. Figure9 measured speed up on increasing threads for two different itemsets, computed from 10% and 20% support values. The scale up is almost constant if the numbers of compute threads involved in computation are same.

We have measured speed up for different datasets as shown in Figure10. It shows that compute time linearly increased with increase in dataset, however it is almost constant if used parallel environment, GPU threads. Figure 11 shows that as the compute threads increases speed up increases. Figure12 shows that if increased data load computed on increasing compute threads then scale up is almost linear. This linear scale up is achieved on shifting of Knee point by providing the additional compute threads. This empirical test proves that, compute time remains almost constant if increasing data size computed with increasing compute threads, hence achieves scalability.

5. CONCLUSION

In our work, we have designed and implemented parallel apriori to make association rules mining scalable, efficient, and speedy process. On implementing the sequential Apriori algorithm, it has been observed that, it is difficult to handle large amount of data for frequent itemset mining with compute efficiency. It has been found that our ParApriori algorithm has linear scaleup on increasing load. We observed that Apriori has good performance potential for multi- and many-core platforms and is the best candidate for parallel implementation. Implementation on many threaded GPU cores shown reasonable increase in scaleup for Apriori algorithm. Our empirical analysis shows that ParApriori is an efficient way to parallelize the frequent itemset mining tasks on heterogeneous computing platform to achieve good scaleup.

ACKNOWLEDGMENTS

The authors of this paper thanks to all the supporters, who has helped directly or indirectly in our research work. Especially, we are very much thankful to Kislay Bhatt, Senior Scientific officer, and Mr. Venkat PPK Scientific Officer from Supercomputing Section and Visualization Section respectively of Computer Division, BARC, for their timely help. We also express thanks to NVIDIA Pune team, especially Ms Jaya Panvelkar, Mr. Pradeep Gupta, and others, for providing support to complete our work.

REFERENCES

[1]. V.B.Nikam, B.B.Meshram, “Scalability Model for Data Mining”, ICIMT2010, Dec 28-30, 2010, 978-1-4244-8882-7/2010, IEEE

[2]. V.B.Nikam, Kiran Joshi, B.B. Meshram, “An Approach for System Scalability for Video on Demand”, International Conference on Network Infrastructure Management Systems (Interface2011), Dec16-17, 2011.

[3]. Kargupta, H.Sivakumar, K. “Existential pleasures of distributed data mining”, In Data Mining: Next Generation Challenges and Future Directions, edited by H. Kargupta, A. Joshi, K. Sivakumar, e Y. Yesha, MIT/ AAAI Press, 2004.

[4]. Park, B.,Kargupta, H., “Distributed Data Mining: Algorithms, Systems, and Applications”, In The Handbook of Data Mining, edited by N. Ye, Lawrence Erlbaum Associates, pp: 341-358, 2003.

[5] .R.Agrawal and H.Mannila, “Fast Discovery of Association Rules”, in Advances in Knowledge Discovery and Data Mining. p. 307-328, 1996.

[6]. R.Agrawal and R.Srikant, “Fast algorithm for mining association rules”, VLDB, p. 487-499, 1994

[7]. M.J.Zaki and K.Gouda, “Fast Vertical Mining Using Diffsets”, Proc. SIGKDD, p. 326-335, 2003.

[8]. J.Han, H.Pei, and Y. Yin., “Mining Frequent Patterns without Candidate Generation”, SIGMOD. , 2000.

[9]. N.Govindaraju and M. Zaki, “Advances in Frequent Itemset Mining Implementations”, FIMI 2003.

[10]. F.Bodon, “A Trie-based APRIORI Implementation for Mining Frequent Item Sequences”, OSDM, 2005.

[11]. C.Borgelt, “Efficient Implementations of Apriori and Eclat”, Proc.FIMI 2003.

[12]. Yanbin Ye, Chia-Chu Chiang, “A Parallel Apriori Algorithm for Frequent Itemsets Mining”, Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications, SERA'06, Pages 87-94, 2006.

[13]. Putte gowda D, Rajesh Shukala, Deepak N A, “Performance Evaluation of Sequential and Parallel Mining of Association Rules using Apriori Algorithms”, Int. J. Advanced Networking and Applications Volume: 02 Issue: 01 Pages: 458-463, 2010.

[14]. R.Agrawal, J.C. Shafer “Parallel Mining of Association Rules” IEEE Transactions on Knowledge and Data Engineering, Volumes 8,Number 6, PP 962-969, December 1996.

[15]. S.arthasarathy, M. J. Zaki, M. Ogihara, W. Li, “Parallel Data Mining for Association Rules on Shared-Memory Systems”, Knowledge and Information Systems, v3:p1-29, 2001.

[16]. Rasmus Resen Amossen, Rasmus Pagh, “A New Data Layout For Set Intersection on GPUs”, Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium,IPDPS'11, P:698-708,2011.

[17]. George Teodoro, Nathan Mariano, Wagner Meira Jr., Renato Ferreira, “Tree Projection-based Frequent Itemset Mining on multi-core CPUs and GPUs”, Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing, P:47-54. SBAC-PAD '10, 2010

[18]. NVIDIA, NVIDIA CUDA SDK, 2009.

[19]. Freitas, A.;Lavington, S. H., “Mining very large databases with parallel processing”, Kluwer Academic Publishers, The Netherlands, 1998.

[20]. William Gropp, Ewing Lusk, “Users Guide for MPICH a portable implementation of MPI Technical reports ANL-96/6”, Argonne National Laboratory 1996.

[21]. Ana C.M.P.de Paula, Bráulio C.Ávila, Edson Scalabrin, FabrícioEnembreck, “Using Distributed Data Mining and Distributed Artificial Intelligence for Knowledge Integration”, Proceedings of the 11th international workshop on Cooperative Information Agents XI, CIA '07, P:89-103, 2007.

[22]. G.Piatetski-Shapiro, “Discovery, Analysis, and Presentation of Strong Rules”, Proceedings of Knowledge Discovery in Databases, Piatetski-Shapiro, G., editor, AAAI/MIT Press, pp.241-248, 1991

[23]. Ferenc Bodon and L.R´onyai. “Trie: an alternative data structure for data mining algorithms”, Computers and Mathematics with Applications, 2003.

[24]. M.Zaki and S.Parthasarathy, “New Algorithms for Fast Discovery of Association Rules”, in KDD. p. 283-296, 1997.

[25]. J.Ruoming, Y.Ge, G. Agrawal, “Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance” IEEE Transactions on Knowledge and Data Engineering, Vol 17, Issue 1, 2005.

[26]. R.Agrawal, and R.Srikant, “Quest Synthetic Data Generator”, IBM Almaden Research Center, San Jose, California,” 2009.

Authors

Valmik B Nikam has done Bachelor of Engineering (Computer Science and Engineering) from Government College of Engineering Aurangabad, Master of Engineering (Computer Engineering) from VJTI, Matunga, Mumbai, Maharashtra state, now pursuing PhD in Computer Department of VJTI. He was faculty at Dr. Babasaheb Ambedkar Technological University, Lonere. He has 12 years of academic experience and 5 years of administrative experience as a Head of Department. He has 1 year of industry experience. He has attended many short term training programs and has been invited for expert lectures in the workshops. Presently he is Associate Professor at Computer Engineering & Information Technology Department of VJTI, Matunga, Mumbai. His research interests include Scalability of Data Mining Algorithms, Parallel Computing, Scalable algorithms, GPU Computing, and Cloud Computing, Data Mining, Data Warehousing. Mr. Nikam is a member of CSI, ACM, IEEE research organizations, and also a life member ISTE. He has been felicitated with IBM-DRONA award in 2011.

B.B.Meshram is a Professor and Head of Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute, Matunga, Mumbai. He is Ph.D. in Computer Engineering. He has been in the academics and research since 20 years. His current research includes database technologies, data mining, securities, forensic analysis, video processing, distributed computing. He has authored over 203 research publications, out of which over 38 publications at National, 91 publications at international conferences, and more than 71 in international journals, also he has filed 8 patents. He has given numerous invited talks at various conferences, workshops, training programs and also served as chair/co-chair for many conferences/workshops in the area of computer science and engineering. The industry demanded M.Tech program on Network Infrastructure Management System, and the International conference "Interface" are his brain childs to interface the industry, academia and researchers. Beyond the researcher, he also runs the Jeman Educational Society to uplift the needy and deprived students of the society, as a responsibility towards the society and hence the Nation.