



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Parallelizing Frequent Itemset Mining Process using High Performance Computing

¹Sheetal Rathi, ²Dr.Chandrashekhar.Dhote

¹PhD Research Scholar, SGBAU, Amravati, Maharashtra, India

²System Manager, PRMITR, Badnera, Maharashtra, India

Abstract: Data is growing at an enormous rate and mining this data is becoming a herculean task. Association Rule mining is one of the important algorithms used in data mining and mining frequent itemset is a crucial step in this process which consumes most of the processing time. Parallelizing the algorithm at various levels of computation will not only speed up the process but will also allow it to handle scalable data. This paper proposes a model to parallelize the frequent itemset mining process without additional load of using multiprocessors using high performance computing. GPU have been used which offer better performance at significantly low cost and are also energy efficient as compared to multi-core multiprocessors.

Keywords: Frequent Itemset Mining, FP-growth, scalability, High Performance computing, GPU

I. Introduction

Since large amount of data is available widely, extracting knowledge from this data is necessitating the researchers to speed up the data mining process. Developing an efficient algorithm that can handle large volumes of data is a big challenge faced today as data is growing at an enormous rate. One of the main areas of data mining is association rule mining and is shown to be useful in major application domains such as biomedical, telecommunication and financial sector.

Many algorithms have been proposed for association rule mining during the years. However the basic two foundation algorithms are Apriori [1] and FP-growth algorithm [2]. Henceforth improvisations were done on these two basic algorithms. Some algorithms adopted an Apriori-inspired method, which try to reduce the time required for scanning databases by focusing mainly on reducing candidates. But major drawback of Apriori based algorithm is that the potential candidates grow at an exponential rate. This will be visibly noticeable if the data is large. Han et al. [2] put forward the Frequent-Pattern-tree structure for efficiently mining association rules without generation of candidate itemsets. To mine large dataset patterns it implemented a divide-and-conquer database projection strategy. Its main merits are: 1) without producing candidates it directly constructs the FP-Tree and condition FP-Tree, and produce frequent pattern by visiting FP-Tree recursively; 2) Only two database scans are required. The experimental results showed that FP-Tree and almost all its extensions work better for dense datasets as compared to Apriori..

The proposed model is based on parallelizing FP-growth algorithm. The highly parallel version of FP – Growth can be implemented on scalable data. Although scalability can be handled using multi-core processors, but it shows poor performance for more if processors are more than eight. Using high performance computing GPU, the model will handle the computational cost of using multi-core processors.

II. Pre-Requisites

A. Frequent Mining Algorithm

The formal statement of frequent mining is as follows [2]:

Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of literals, called items and n is considered the dimensionality of the problem. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A transaction T is said to contain X , a set of items in I , if $X \subseteq T$. An itemset X is said to be frequent if its support is greater than or equal to a given minimum support threshold specified by the user.

The association rule mining is a two-step process:

- 1) Find all frequent itemset having minimum support.
- 2) Generate strong rules having minimum confidence, from the frequent itemset.

FP-Growth uses a pattern fragment growth method to generate frequent itemsets. It requires only two scans. The first is to collect the support of each item and then put the frequent items into a list, F-List, sorted in descending order of frequency. The original database is converted into a compressed tree structure, the FP-tree in the second scan. Once a FP-tree is constructed, the FP-Growth works in a divide and conquer approach, decomposing the FP-tree into a group of independent conditional pattern bases of the frequent items. FP-Growth needs to build conditional FP-tree according to each conditional pattern base. Henceforth, the mining process is conducted on independent processes, constructing and mining conditional FP-tree recursively. The pseudo code of mining on FP-tree is depicted in Figure1.

Figure 1 Pseudo code of FP-Growth algorithm from [3]

```

procedure FP-Growth (Tree,  $\alpha$ )
{
  if Tree contains a single path  $P$  then
    for each  $\beta$ = nodes combination in  $P$  do
      pattern =  $\beta \cup \alpha$  ;
      support = min(support of the nodes in  $\beta$ );
    else
      for each  $ai$  in the header of Tree do
        pattern  $\beta = ai \cup \alpha$ ;
        with support =  $ai$ . support ;
        construct conditional pattern base of  $\beta$ 
        TreeB = construct conditional FP-tree of  $\beta$ 
        if TreeB  $\neq \emptyset$  then
          call FP-Growth( TreeB ,  $\beta$ )

```

B. High Performance Computing

CUDA (Compute Unified Device Architecture) [3] enables users to write scalable multi-threaded programs for CUDA- enabled GPUs which suffice the purpose of handling scalable data. Because of the hardware embedded multithreading, the GPU can easily take care of compute-intensive portions of the application while the remaining code can be handled by the CPU. It is a parallel computing architecture that is used to solve many complex computational problems in a fraction of the time required on a CPU. In short, a GPU is a coprocessor to the CPU with its own DRAM and runs many threads in parallel. Fig 2 shows the basic architectural difference between a CPU and GPU which supports the purpose of using a GPU to handle scalable data.

Figure 2 Architecture difference between CPU and GPU[3]



The CUDA Instruction Set Architecture and the parallel compute engine in the GPU are the two main constituents of CUDA. It performs two major functions: it helps the CPU usage by engaging the GPU's stream processors and any computing process where CUDA is enabled can be accelerated. Kernel is the sequential part of CUDA programming. The various operations are performed on threads and are invoked as a set of concurrently executing threads. These threads are organized in a hierarchy consisting of thread blocks and grids. Fig.3 shows the computational power of NVIDIA GPUs (Graphics Processing Units) of GeForce 8800 which greatly reduce processing times for connectivity mapping.

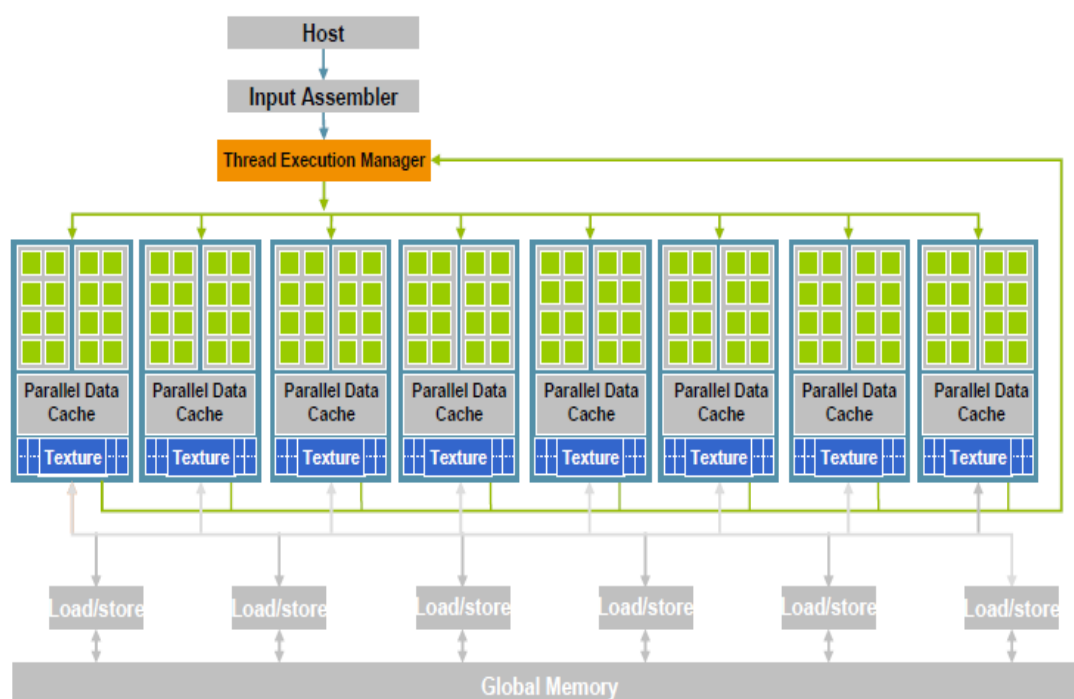
III. Related Work

Data parallelism deal with lot of challenges in terms of load balancing, costs of communication and synchronization. Most parallel data mining algorithm were developed from Apriori based method. However it was shown that FP-growth based algorithm require only two database scans and is an order of magnitude faster than Apriori algorithm. So attempts were done to parallelize FP-growth. But although FP-growth is faster than Apriori algorithm it still has an issue with memory consumption as tree generation requires lot of memory space and as the data grows in size, it is not possible to fit the entire tree in main memory.

Aouad et al. [4] designed a distributed Apriori on heterogeneous computer clusters and grid environments. To tackle memory constraint, dynamic workload management has been used and they have achieved balanced workloads and thus were able to reduce the communication costs. In [5] a linear speedup was achieved by Li et

al. for the FP-Growth algorithm over thousands of distributed machines. They have made use of Google's MapReduce infrastructure. Buehrer [6] proposed variants of FP-Growth on computer clusters and have tried lowering communication costs and improve the cache memory and I/O utilization. Min Chen et.al [7] proposed a novel parallel FP-Growth algorithm GFP-Growth. Memory Overflow is avoided by running it on computer cluster. Using projection method and splitting the mining task into number of independent sub-tasks, this algorithm works independently at each node. As a result, it can efficiently reduce the inter-node communication cost. In [8], Wenbin Fang et al. introduced GPUMiner, a novel parallel data mining system. It utilizes new-generation graphics processing units (GPUs). The authors have tried to parallelize Apriori algorithm. Their system relies on the massively multi-threaded SIMD (Single Instruction, Multiple-Data) architecture provided by GPUs. We are trying to implement a similar approach using GPU architecture to parallelize FP-growth algorithm. Our model is proposed to parallelize frequent mining process based on the guidelines of FP-growth but without generating FP tree.

Figure 3 Block Diagram of G80 CUDA Mode [3]



IV. Proposed Model

Although because of its irregular data structure and complex algorithmic control, FP-growth poses a great challenge for GPU acceleration, we propose a model which is using the high performance power of a GPU and parallelize FP-growth at various levels. The major advantages of FP growth algorithm as stated above are:

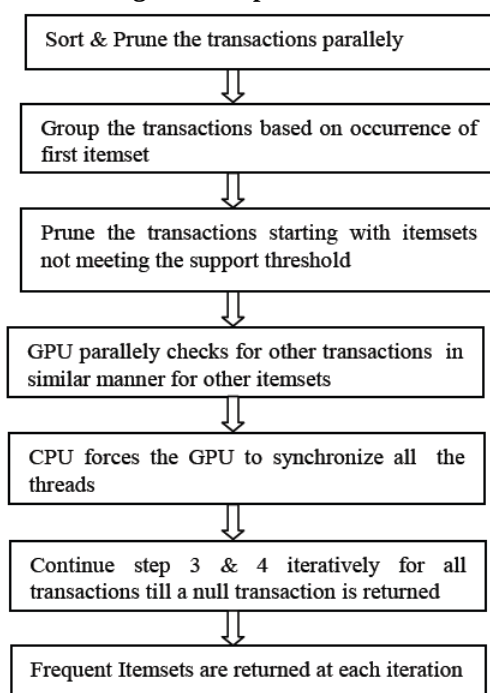
1. Less number of database scans/reads
2. Less number of permutations to consider while checking for patterns.

However, to achieve this, a tree is to be constructed which may tend to grow extremely huge in size. Moreover, traversing this tree will require some major computations (although comparatively much less than Apriori approach). Many approaches have already been proposed to parallelize this. The two primary concepts on which these approaches are based being:

1. Build a huge tree parallelly and then traverse from the leaf upwards in parallel.
2. Build several small FP trees in parallel and then find patterns in each. After this, merge the patterns obtained by a predefined metric/mechanism.

Because of the herculean task of building and traversing the FP tree, we propose a model wherein we make use of the FP growth algorithm's advantages while eliminating the tree construction phase. The aim will be to save time required to build the tree and to reduce the pattern search time in the form of arrays.

Figure 4 Proposed model



It can be analyzed from the various mining algorithms that preprocessing the dataset appropriately and efficiently generates considerable better and faster results [9]. The sorting process which is parallelized is using selection sort algorithm. The following example will explain the concept of frequent itemset mining using parallelization. Table 1 shows an unordered list of 10 transactions with 8 unique itemsets.

Table 1 Transaction Database

Transaction no.	Itemsets
1	a,b,c,d,e
2	c,f,a
3	c ,b, e
4	d ,a, f, b, c, e
5	a ,b, d, f
6	g ,e, a, b
7	h ,f
8	c ,d, a, b
9	g ,h, e, f, b, c
10	b,d,c,f

In the first step we sort the transactions parallelly in pre-processing stage so as to make the dataset favourable for mining. The file contains randomly generated characters between ASCII values 65 to 122 with each line having a random number of characters. The ASCII value is added to the product of support count of each character and 122 i.e. ASCII value+ (Support count of char*122). The transactions are read and stored in fixed size 2D array with enough columns to store the longest transaction.

All the transactions are read first and a unique character list is made containing all the characters used in all the transactions along with the count of how many times it occurs. Those characters having a count less than that of the minimum support are eliminated from this list and their count is also removed. The 2D character array is then converted into an integer 2D array where the characters present in the unique array are converted to their ASCII values and those that are not present are given a value of 150. This 2D array is then flattened to a 1D array with the index of the 1D array going from 0 to 99. This flattened array is then passed to the GPU for sorting. We sort in the ascending order. The original FP algorithm sorts in descending order but we do it in ascending order to better utilize our algorithm. Each GPU block process different parts of the 1D array which correspond to individual rows in the original 2D array.

In the next step, after the 1D array is sorted and converted back to characters, we send it back to the GPU for the FP algorithm. Each block again handles parts of the 1D array which originally represented a row. The sorted 2D array looks as follows:

Table 2 Sorted 2D Array

e	a	c	b	-	-	-	-	-	-
a	f	c	-	-	-	-	-	-	-
e	c	b	-	-	-	-	-	-	-
e	a	f	c	b	-	-	-	-	-
a	f	b	-	-	-	-	-	-	-
e	a	b	-	-	-	-	-	-	-
f	-	-	-	-	-	-	-	-	-
a	c	b	-	-	-	-	-	-	-
e	f	c	b	-	-	-	-	-	-
f	c	b	-	-	-	-	-	-	-

The above step is followed by sending the sorted 2D array back to the GPU for parallelizing FP algorithm. The GPU then groups the transactions based on the first character that occurs. The grouping happens as follows:

Table 3 Grouping for transactions starting with ‘f’

f	-	-	-	-	-	-	-	-	-
f	c	b	-	-	-	-	-	-	-

Table 4 Grouping for transactions starting with ‘a’

a	f	c	-	-	-	-	-	-	-
a	f	b	-	-	-	-	-	-	-
a	c	b	-	-	-	-	-	-	-

Table 5 Grouping for transactions starting with ‘e’

e	a	c	b	-	-	-	-	-	-
e	c	b	-	-	-	-	-	-	-
e	a	f	c	b	-	-	-	-	-
e	a	b	-	-	-	-	-	-	-
e	f	c	b	-	-	-	-	-	-

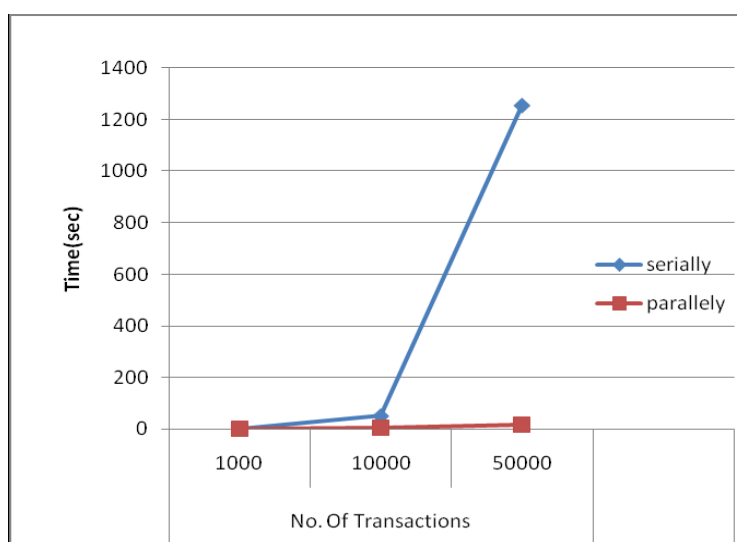
The least frequent element occurring (<min support) is ‘f’. Those transactions that start with ‘f’ are grouped together. The count of the number of times it occurs in all the transactions is checked and if it is less than the threshold that is set, the itemset is eliminated and sent back to the pool of transactions.

Here the transactions starting with ‘f’ does not satisfy the threshold requirement. However, ‘a’ and ‘e’ satisfy the threshold requirements. Hence ‘f’ is eliminated and those transactions are sent back to the pool. However, if on removing the element from the transaction we find that the transaction becomes void then that transaction is simply discarded. The other transactions are then checked and eliminated in parallel in a fashion similar to ‘f’. The CPU forces the GPU to synchronize all threads after step 3 & 4 since some threads will take longer than expected to compute because of lengthy transaction. If a pattern meets the threshold requirement, it is recorded and further processing is continued on it. If on further processing no pattern is found frequent, the processing is stopped for that transaction. This iterative process is carried out parallelly in the GPU.

V. Result and Conclusion

The above model is being developed using AMD Athlon 64 X2 Dual Core Processor 4000+ with 2 GB RAM on an Nvidia GT 520 (1 GB) GPU. The first step of sorting and pruning the transactions in preprocessing is tested on a set of 1K, 10K and 50K synthetic transactions respectively. Figure 5 shows the time comparison for parallel and serial sorting which was done on the synthetic dataset. It shows that that the processing speeds up remarkably when done parallelly.

Figure 5 Time comparison of serial and parallel selection sort



Handling scalable data is a crucial issue in mining frequent itemset. Parallelization is one of the best alternatives for handling big data. Although multi processors can be used for parallelization, they pose additional burden in terms of cost and a big experimental setup as well as can work efficiently only up to 8 processors. We propose a unique model wherein we make efficient use of low cost and energy efficient GPUs. We try to parallelize the frequent mining process by working on the model of FP growth but without generating FP tree with the help of arrays. The first step of parallelizing the pre-processing is done with significant speedups as compared to its sequential counterpart. In future, we will try to extend the parallelization process at various steps of our proposed algorithm. Bitonic sort can also be incorporated instead of selection sort.

VI. References

- [1] R. Agrawal, T. Imielinski, and A. Swami. 1993. Mining association rules between sets of items in large databases. In: ACM SIGMOD Intl. Conf. Management of Data. S. M. Metev and V. P. Veiko, Laser Assisted Microtechnology, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.
- [2] Han J, Pei J, Yin Y. 2000. Mining frequent patterns without candidate generation. In Proc. of the ACM SIGMOD Conference on Management of Data. Dallas, TX.
- [3] <https://developer.nvidia.com/cuda-gpus>.
- [4] L. Aouad, N. Le-Khac, and T. Kechadi. 2007. Distributed Frequent Itemsets Mining in Heterogeneous Platforms. 2007. In Journal Of Engineering, Computing and Architecture, Vol.1 Issue 2. ISSN 1934-7197.
- [5] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang. 2008. PFP: Parallel FP-Growth for Query Recommendation. In ACM Recommender System.
- [6] G. Buehrer, S. Parthasarathy, S. Tatikonda, T. Kurc and J. Saltz. 2007. Toward terabyte pattern mining: an architecture conscious solution. In PPOPP, 2007.
- [7] Min Chen, XueDong Gao HuiFei Li, "An Efficient Parallel FP-Growth Algorithm", 978-1-4244-5219-4/09, 2009 IEEE
Wenbin Fang, Ka Keung Lau, Mian Lu, Xiangye Xiao, Chi Ki Philip Yang Yang, Bingsheng He1, Qiong Luo, Pedro V. Sander, and Ke Yang. 2008. Parallel Data Mining on Graphics Processors. In Technical Report HKUSTCS0807, Oct 2008.