I.J. Modern Education and Computer Science, **2021, 4, 28-41**
Published Online August 2021 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2021.04.03

GNVDF: A GPU-accelerated Novel Algorithm for Finding Frequent Patterns Using Vertical Data Format Approach and Jagged Array

P. Sumathi
Research Scholar, Nehru Memorial College (Affiliated to Bharathidasan University), Puthanampatti, Tiruchirappalli-Dt, Tamil Nadu, India - 621007
Email:sumiparasu@gmail.com

S.Murugan
Associate Professor, Nehru Memorial College (Affiliated to Bharathidasan University), Puthanampatti, Tiruchirappalli-Dt, Tamil Nadu, India - 621007
Email:murugan_nmc@hotmail.com

Received: 01 June 2021; Revised: 28 June 2021; Accepted: 24 July 2021; Published: 08 August 2021

Abstract: In the modern digital world, online shopping becomes essential in human lives. Online shopping stores like Amazon show up the "Frequently Bought Together" for their customers in their portal to increase sales. Discovering frequent patterns is a fundamental task in Data Mining that find the frequently bought items together. Many transactional data were collected every day, and finding frequent itemsets from the massive datasets using the classical algorithms requires more processing time and I/O cost. A GPU accelerated Novel algorithm for finding the frequent patterns using Vertical Data Format (GNVDF) has been introduced in this research article. It uses a novel pattern formation. In this, the candidate *i*-itemsets is divided into two buckets viz., Bucket-1 and Bucket-2. Bucket-1 contain all the possible items to form candidate-(i+1) itemsets. Bucket-2 has the items that cannot include in the candidate-(i+1) itemsets. It compactly employs a jagged array to minimize the memory requirement and remove common transactions among the frequent 1-itemsets. It also utilizes a vertical representation of data for efficiently extracting the frequent itemsets by scanning the database only once. Further, it is GPU-accelerated for speeding up the execution of the algorithm. The proposed algorithm was implemented with and without GPU usage and compared. The comparison result revealed that GNVDF with GPU acceleration is faster by 90 to 135 times than the method without GPU.

Index Terms: Frequent Patterns, GNVDF, Graphical Processing Unit, Novel Pattern Formation, Vertical Data Format, and Jagged Array.

1. Introduction

Data Mining (DM) is a part of Knowledge Discovery in Databases (KDD) [1] and explores the hidden patterns for business people. It is associated with many fields such as database systems, data warehousing, statistics, machine learning, information retrieval, and high-level computing [2,3]. It is also supported by other sciences like neural networks, pattern recognition, spatial data analysis, image databases and signal processing [2,3]. There are several techniques in data mining like classification, clustering, association rule mining and regression [4]. Frequent Pattern Mining (FPM) is a computationally crucial step in data mining [5]. It is used to determine the frequent patterns and associations from databases such as relational and transactional databases and other data repositories. The Apriori is one of the most important algorithms for finding frequent itemsets. It has many problems such as more database scan and I/O cost, a large amount of time etc., for finding frequent itemsets. So the researchers have made several refinements to Apriori in the last two decades.

However, enhancing speed and reducing memory requirements are the essential parameters while determining the frequent patterns nowadays because of the rise of big data in various domains and sources in human endeavour. Also, when the transactional database size increases, demand for storage is increased and requires high-speed algorithms to find frequent patterns. But with a single-threaded approach, it's tough to minimize time. The GPU accelerated computing employs GPUs along with CPUs. It enables superior performance by supporting a parallel programming paradigm with multiple cores. It saves time and cost in scientific and other high computing tasks [6]. So, researchers

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

were utilized GPUs in FPM based research. Some research works based on GPUs that motivate this article's proposed work were discussed here.

W. Fang et al. [7] have introduced two implementations for Apriori using GPUs with Single Instruction, Multiple Data (SIMD) architectures. Both methods use a bitmap data structure. They executed the first one on the GPU, avoiding the intermediate data transfer between the GPU and CPU memory. The second one uses both the CPU and GPU for processing with trie structure. They proved that both implementations speed up the processing than the classical Apriori algorithm. S. M. Fakhrahmad et al. [8] have developed different parallel versions of a novel sequential mining algorithm for finding frequent itemsets. The methods are i) assigning each partition to a processor, ii) assigning each column to a processor, and iii) devoting the k^{th} processor to mine the k^{th}-itemsets. These methods were compared experimentally using time complexity, communication rate, and load balancing and proved that the proposed methods outperformed the existing sequential algorithms.

The authors J. Zhou et al. have designed [9] a GPU-based Apriori algorithm with OpenGL to accelerate association rules mining. The experiment proved that the proposed algorithm provides better performance than the classical algorithms. A new pattern-based algorithm called HSApriori has been suggested by D. William Albert et al. [10], and it is based on the parallel processing nature of GPU. In this, the proposed method was tested using both the tidset and bitset representation of the dataset and found that the bitset is more appropriate for parallel processing. Further, they proved from the experiment that the speed of HSApriori is substantially more when compared with traditional HorgeltAprirori.

To solve the limitations of Apriori, a parallel Apriori Map Reduce model has been presented by M. Tiwary et al. [11] using high-performance GPU. They have attached a GPU with every node in a Hadoop cluster. Also, they have used NVIDIA's GPU and JCUDA and JNI for the integration process. From the experiments, it has been proved that it provides better performance in terms of execution time. The downside of the algorithm is that the extra hardware charge is associated with the GPUs in each node in the Hadoop cluster. To overcome the drawbacks in the traditional cluster-based map-reduce, J. Li et al. [12] have designed a multi-GPU based parallel Apriori algorithm to accelerate the calculation process of Apriori. It has been initiated especially to mine association rules in medical data. The analytical results have proved that the proposed method significantly improves the execution speed with a lower cost for medical data.

A novel method called CGMM to suit both sparse and dense datasets has been proposed to mine frequent patterns has been introduced by L. Vu et al. [13]. To increase the speed of the FPM process, it combines both the CPU and GPU. In this method, the CPU uses the FP-tree data structure to perform mining, and the GPU converts the data to bit vectors. The experiments with AMD CPUs and NVIDIA GPU have proved that the performance evaluation of CGMM is faster than the existing sequential FPM and GPApriori. Y. Li et al. [14] have developed a GPU-based algorithm called Multi-level Vertical Closed FIM. In this, a multi-layer vertical data structure has been used to minimize the usage of storage. The implementation is being accelerated with GPU to achieve high-speed computation, mainly on large and sparse datasets.

K.W. Chon et al. [15] have proposed a novel algorithm called GMiner. It is a GPU-based method for finding frequent itemsets on large-scale datasets. It determines the patterns from the first level of the enumeration tree rather than storing and utilizing the patterns at the intermediate levels of the tree. With the computational power of GPUs, the method achieved fast performance and outperformed significantly than the existing sequential and parallel methods. The method also eliminates the skewness problem that the parallel algorithms suffer. A Dynamic Queue and Deep Parallel (D2P) Apriori algorithm was generated by Y. Wang et al. in [16]. In this, the candidate generation process has been parallelized by using the Graph-join and dynamic bitmap queue. It also uses a vertical bitmap structure with low-latency memory on GPU. The experiments have explored that the D2P-Apriori obtained high-speed up, i.e. a 23×speed up ratio compared to the modern CPU methods.

The authors Y. Djenouri et al. [17] have created three High-Performance Computing (HPC)-based versions of Single Scan (SS) for frequent itemset mining viz., GSS, CSS, and CGSS. The GSS, CSS, and CGSS implement SS with GPU, cluster architecture, and GPU with multiple cluster nodes. They have also presented three approaches to reduce cluster load balancing and GPU thread divergence. The experiments have proved that the CGSS performs best in speed than SS, GSS and CSS.

The authors P.Sumathi et al. [18] have developed a memory-efficient implementation for a vertical data format approach in finding frequent patterns using jagged array matrix representation. They have formulated mathematical equations for memory requirements and proved that it reduces the memory requirement than the traditional multidimensional array.

The numerous GPU based FPM algorithms found in the literature have their own merits. But they have some performance, data size and scalability issues [19], which provides a more vital lead to the proposed work. The research article has introduced GNVDF, a novel GPU-accelerated FPM algorithm. It uses a novel pattern generation method to avoid generating many candidate itemsets as classical algorithms and uses a compact jagged array structure to minimize storage space [18]. Further, it uses the VDF format of transactional data to reduce the number of disk accesses.

The remaining paper is organized as follows. Section 2 presents the basic terminologies and definitions, vertical data format, jagged array, and GPU. The description of the proposed methodology with an illustration is presented in

section 3. Section 4 illustrates the experimental results and discussion. Finally, the research article ends with a conclusion in section 5.

2. Basic Concepts

Finding frequent itemsets is essential in mining associations, correlations, and many other relationships among the data. It is used in data classification, clustering, and other data mining tasks. Thus, FPM is focused on data mining research, and this section briefs the fundamental concepts associated with FPM and the study.

A. Basic Terminology

An itemset (set of items) that contains k items is said to be a k-itemset. The set of laptop, printer is a 2-itemset. Frequent patterns are the patterns (itemsets, subsequences, or substructures) that frequently appear in a dataset [2,20]. The support count of the itemset is identified by the number of transactions that contain the itemset. A sequence is an ordered list of itemsets, i.e. set of items purchased together. A subsequence is a sequence of items bought together and frequently occurs in a transactional database known as a sequential pattern. A substructure can be represented in different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences [2].

B. Basic Definitions

Let $I=\{I_1, I_2,\ldots, I_m\}$ be an itemset, and D is a transaction database contains a set of transactions T is a non-empty itemset such that $T \subseteq I$ and each transaction T is associated with a unique identifier TID. Let A be a set of items. A transaction T is said to contain in A if $A \subseteq T$. The format of the association rule is $A \rightarrow B$, where $A \subset I$, $B \subset I$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \emptyset$ [21]. Associations rule $A \rightarrow B$ that holds in the transaction database D with support (s) and confidence(c) [1].

Support(s): The support of an association rule $A \rightarrow B$ is defined as the percentage of records that contain $A \cup B$ to the total number of records in the database [22]. It is noted that the support count is increased when an item present in numerous transactions in the database D [22].

Confidence: The confidence of a rule $A \rightarrow B$ is defined as $s(A \rightarrow B)/s(A)$. It is the ratio of the number of transactions that contain all items in the consequent (B), as well as the antecedent (A) to the number of transactions that include all items in the antecedent (A) [23].

The minimum support threshold is used to discover the frequent itemsets from the databases. In contrast, the minimum confidence constraint is applied to those frequent itemsets found previously in determining the best rules.

C. Vertical Data Format

The databases can be represented in FPM algorithms in two data formats. They are i) Horizontal Data Format (HDF) and ii) Vertical Data Format (VDF). HDF represents the items categorized into particular transactions as stored in the database. i.e. it is denoted as <TID, Itemset>, where TID is the transaction ID, and Itemset refers to the items purchased by the customer corresponding to TID. The VDF represents data as transactions categorized into particular items that mean the TIDs are grouped for each item, i.e. VDF is described by <Item, Tid_set>, where item denotes an item in the shop and Tid_set contains the TID's where the item occurs. Fig.1. and Fig.2. show the HDF and VDF of D.

TID	Itemset
0:	{c,d,e,g,h,i,k,p,m}
1:	{b,e,f,g,h,i,p,m}
2:	{c,e,m}
3:	{a,b,c,d,e,f,g,i,p}
4:	{a,b,c,d,e,p}
5:	{a,b,c,d,f,h,p}
6:	{b,e,f,h,i,p,m}
7:	{a,c,d,e,k,p,m}
8:	{a,c,d,e,f,i,p,m}
9:	{a,c,d,e,f,h,i,p,m}

Fig. 1. HDF of Transaction Database D

D. Jagged Array

A jagged array data structure is an array whose elements are arrays known as "array of arrays" with varying columns in each array/row, and it is shown in Fig.3.

E. Graphical Processing Unit

It is a device specifically designed for graphics processing. It is widely used in large scale hashing and matrix computations because it supports parallelism and serves as the base for mining and machine learning. CUDA and OpenCL are two popular GPGPU programming framework tools. NVIDIA has designed a parallel computing platform and programming called Compute Unified Device Architecture (CUDA) [12,24]. The CUDA-based program can only be run on the NVIDIA-produced GPU. A typical CPU may contain four or eight cores; an NVIDIA GPU consists of thousands of CUDA cores and a pipeline that supports parallel processing on thousands of threads, increasing the speed significantly.

With Numba, the python developer can quickly enter into GPU-accelerated computing. It makes use of both GPU and CPU to facilitate processing-intensive operations viz., deep learning, analytics, and engineering applications. The CUDA Python and Numba help to enhance the speed by targeting both CPUs and NVIDIA GPUs. With this advantage of CUDA python and Numba, the implementation of this proposed work will be GPU accelerated.

Item	Tid_set
a:	{3,4,5,7,8,9}
b:	{1,3,4,5,6}
c:	{0,2,3,4,5,7,8,9}
d:	{0,3,4,5,7,8,9}
f:	{1,3,5,6,8,9}
g:	{0,1,3}
h:	{0,1,5,6,9}
i:	{0,1,3,6,8,9}
k:	{0,7}
m:	{0,1,2,6,7,8,9}
p:	{0,1,3,4,5,6,7,8,9}

Fig. 2. VDF of Transaction Database D

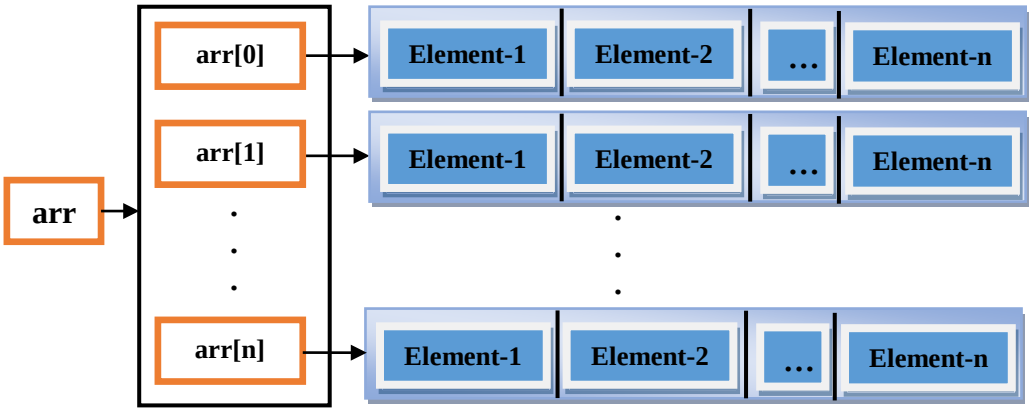

Fig. 3. Jagged array representation

3. Proposed Methodology

The main objective of the proposed work is to find the essential frequent itemsets from the transaction database with less memory space and time by ignoring the least probable ones. The method used Jagged array storage structure [16] and GPU to minimize memory usage and execution time. The proposed method first removes the null/void transactions in the dataset. Null/void transactions are those which contain only one item. Then the dataset is scanned once and converted into VDF format. The support count (*SC*) for each item is calculated by counting the number of transactions that contain each item. Now the candidate 1-itemset C_1 is formed. Next, the frequent 1-itemset is formed by removing the items whose *SC*<min_sup(δ) and stored it in Jagged array representation [18] in sorted order based on *SC*. From L_1 the common transactions among all items are determined either by intersecting or ANDing the transaction in each item, and it is preserved in the Common Transaction List (C_{TID_list}). The transactions in C_{TID_list}'s are removed from each item in L_1, forming the final frequent 1-itemset. The *SC* for each item in L_1 is updated by *SC* - *n*, where *n* is the number of transactions in C_{TID_list}. Next, the new min_sup (δ_{new}) is determined as $\delta_{new} = \delta - n$, and it will be the min_sup from the 2nd iteration onwards.

Before finding the frequent 2-itemset, the final frequent 1-itemset is divided into two logical buckets, LB_1 and LB_2, respectively. LB_1 contains all the items whose $SC = \delta_{new,}$ and the rest will be placed LB_2. The itemset combinations among the items in LB_1 are least probable of being a candidate 2-itemset because the SC of each item is equal to δ_{new}. So it is not considered for generating candidate 2-itemset. The candidate 2-itemsets patterns are generated by combining each item I_x in LB_1 with each item I_y in LB_2 and each item I_z in LB_2 with I_{z+1} in LB_2 until the last item in LB_2. The itemset combination that ends with the last item in LB2 will be placed in C2_2 and the rest in C_{2_1}. From C_{2_1} and C_{2_2}, the items whose SC below the δ_{new} is removed as infrequent and formed L_{2_1} and L_{2_2}.

For generating candidate 3-itemset, each itemset I_x in L_{2_1} is combined with the next item Iy in LB_2 after the last item in I_x. Similar to the previous iteration, the combinations that end with the last item in LB2 are placed in C_{3_2} and rest in $C_{3\_1.}$ It is noted that the itemset combinations in L_{2_2} are not used in the formation of candidate 3-itemsets. The L_{3_1} and L_{3_2} were formed by removing the infrequent itemsets in C_{3_1} and C_{3_2}. The process is continued until L_{n_1} is not null. Further, to increase the execution speed of the proposed method, it is being accelerated with GPU. The proposed algorithm (Algorithm 1) is shown below, and the workflow diagram is shown in Fig.4.

Algorithm 1 A**lgorithm for finding frequent itemsets**

Input : D - a dataset with n transactions;
δ - minimum support threshold;
Output : Frequent patterns;

1: D ← eliminate_null(D);
2: vdf ← scan D and convert it in vertical data format;
3: L_1← one_frequent_itemset(vdf, δ);
4: C_{TID_list} ← find_common_TID(L_1);
5: L_1← remove the transactions in C_{TID_list} for each item in L_1;
6: δ_{new} ← δ - number of transactions in C_{TID_list};
7: LB_1 ← {∀frequent 1-itemset | SC=δ_{new} };
8: LB_2 ← {∀frequent 1-itemset | SC > δ_{new}};
9: L_{2_1}, L_{2_2} ← find_two_freq_itemset(LB_1,LB_2,δ_{new});
10: i=2;
11: while $L_{i\_1} \neq Ø$ do
12: L_{i+1_1},L_{i+1_2} ← n_frequent_itemset(L_{i_1},LB_2,δ_{new});
13: i=i+1;
14: end while

procedure eliminate_null(D - a dataset with *n* transactions)

1: for each $T_i \in D$ do
2: cnt←count the number of items in T_i;
3: if cnt == 1 then
4: remove T_i from D;
5: end if;
6: end for;
7: return D;

procedure one_frequent_itemset(D: Dataset after removing null transactions; δ :minimum support threshold)

1: L_1←Ø;
2: for each $item_i$ in D do
3: TID_{list}←transactions in which $item_i$ occurs;
4: SC←count the number of transactions in TID_{list}
5: if SC ≥ δ then
6: add {$item_i$, TID_{list}, SC}into L_1;
7: end if
8: end for
9: sort L_1 and store it in jagged array format;
10: return L_1;

procedure find_common_TID (L_1: frequent 1-itemset)

1: n←find the number of items in L_1;
2: C_{TID_list}←{TID_{list1} ∩ TID_{list2} ∩… ∩ TID_{listn}};
3: return C_{TID_list};

procedure two_freq_itemset (LB_1: frequent 1-itemset1, LB_2: frequent 1-itemset2, δ:minimum support)

```
1:  last_item←find last item in LB2;
2:  for each itemi in LB1 do
3:      for each itemj in LB2 do
4:          new_pattern ← <itemiitemj>;
5:          new_tid←TIDs(itemi)∩TIDs(itemj);
6:          new_sc←count the transactions in new_tid;
7:          if new_pattern contains last_item then
8:              append{new_pattern,new_tid,new_sc} in C2_2;
9:          else
10:             append{new_pattern,new_tid,new_sc} in C2_1;
11:         end if
12:     end for
13: end for
14: L2_1←{C2_1 | SC(C2_1) ≥ δ};
15: L2_2←{C2_2 | SC(C2_2) ≥ δ};
16: return L2_1, L2_2
```

procedure n_frequent_itemset(L_{i_1}: frequent i-itemset1, LB_2: frequent 1-itemset2, δ_{new}: minimum support)

```
1:  for each itemi in Li_1 do
2:      last_item←find the last item in itemi;
3:      for each itemj in LB2 after last_item do
4:          new_item←{<itemiitemj>};
5:          new_tid←TIDs(itemi)∩ TIDs(itemj);
6:          new_sc←count the transactions in new_tid;
7:          if new_item contains last element in LB2 then
8:              append{new_item,new_tid,new_sc}in Cn_2;
9:          else
10:             append{new_item,new_tid,new_sc}in Cn_1;
11:         end if
12:     end for
13: end for
14: Ln_1←{Cn_1 | SC(Cn_1) ≥ δ};
15: Ln_2←{Cn_2 | SC(Cn_2) ≥ δ};
16: return Ln_1,Ln_2
```

The main advantage of the proposed method is that it reduces the number of candidate itemsets to be generated in each iteration because the itemsets in L_{i_2}, for $i \geq 3$ will not be considered for creating candidate itemsets and removal of items in *CTL* in final L_1. Additionally, GPU and Jagged array enhance the performance in terms of speed and usage of memory.

A. Memory Requirement Calculation

From [25,18], it was observed that the memory requirement using a jagged array structure for the frequent itemsets could be calculated based on the following equation.

$$TM = \sum_{i=1}^{itemset_i \neq \phi} TM_i - rbytes_i \tag{1}$$

where, TM_i is the total memory required for the candidate *i*-itemset, and $rbytes_i$ is the memory occupied by the infrequent/rare items in the candidate *i*-itemset. By subtracting $rbytes_i$ from TM_i, the memory for L_i i.e., frequent *i*-itemsets can be found.

TM_i and $rbytes_i$ were calculated using equations 2 and 3, respectively.

$$TM_i = \sum_{\forall item \in \{itemset_i\}} SC_{item} \times sizeof(tid) + sizeof(item) \tag{2}$$

$$rbytes_i = \sum_{\forall item \in \{in-frequent_i\}} SC_{item} \times sizeof(tid) + sizeof(item) \tag{3}$$

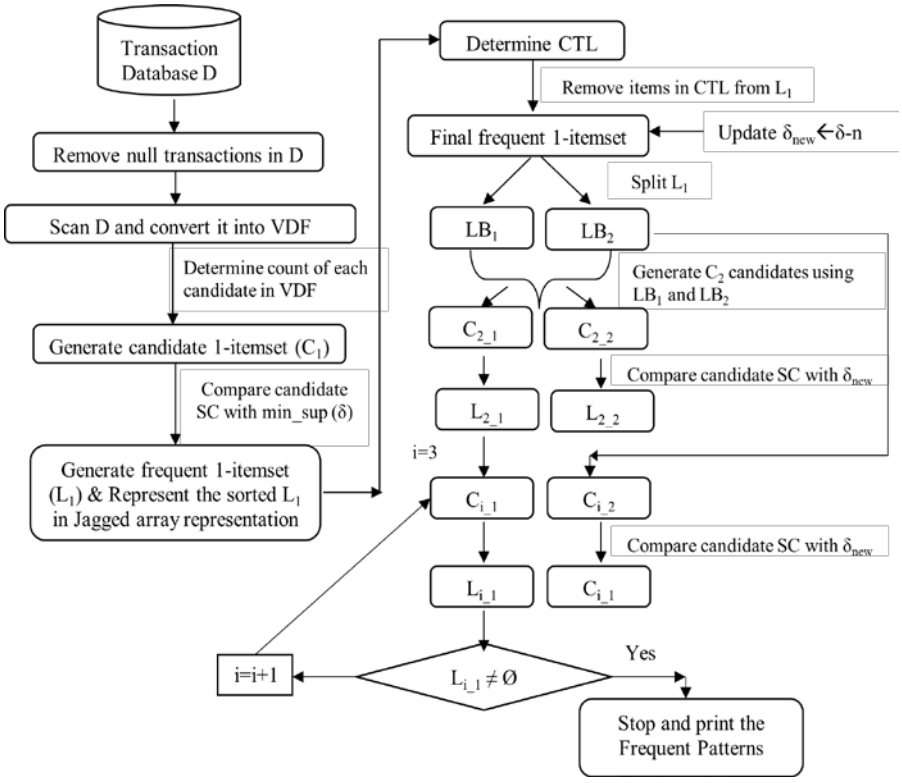

Fig. 4. Workflow of GNVDF

As in [25], the GNVDF also used the same jagged storage structure for storing frequent itemsets, and the amount of memory requirement was calculated as follows. It first fetches the common transactions among items in the frequent 1-itemsets and then removes them from frequent 1-itemsets. Suppose if the frequent 1-itemset contains *n* items say $item_1$, $item_2$, $item_3$,…, $item_n$ and the corresponding *TID* lists say $TID\text{-}List_1$, $TID\text{-}List_2$, $TID\text{-}List_3$,…,$TID\text{-}List_n$, then the common *TID*s(C_{TID}) among the *n* items were found by set intersection operation using equation (4) shown below.

$$C_{TID} = \{TID - List_1\} \cap \{TID - List_2\} \cap ... \cap \{TID - List_n\} \tag{4}$$

The memory space required for C_{TID} was calculated using equation (5).

$$CM = \sum_{i=1}^{length(C_{TID})} sizeof(C_{TID_i}) \tag{5}$$

Since the method removes the C_{TID} from frequent 1-itemsets, the C_{TID} need not be repeated in the subsequent frequent itemsets, saving memory space considerably. The amount of memory saved (*MS*) for the entire dataset was calculated using equation (6).

$$MS = count(itemset_i) \times CM + \sum_{i=2}^{itemset_i \neq \phi} \{count(itemset_{i-1}) + count(itemset_{i-2})\} \times CM \tag{6}$$

where, $count(itemset_1)$, $count(itemset_{i\_1})$, and $count(itemset_{i\_2})$ refer to the number of items in frequent 1-itemset, first and the second part of frequent *i*-itemsets, respectively. Thus, the total memory required for the frequent itemsets of the entire dataset using the proposed method was calculated using equation (7).

$$TM_{final} = \{ \sum_{i=1}^{itemset_i \neq \phi} TM_i - rbytes_i \} - MS \quad (7)$$

B. Proposed Methodology: An Example

The vertical representation of transaction dataset D as shown in Fig. 2 is considered to understand the proposed methodology. It contains 12 items viz., {a, b, c, d, e, f, g, h, i, k, m, p}. Each item is represented by a row containing the name of the item and the transactions in which the item occurs (TIDs) [26]. Let δ is 6. From Fig. 2, the candidate 1-itemset is calculated. The candidate 1-itemset contains all the items in D, the TIDs in which the item occurs and the SC. It is shown in Table 1.

Table 1. Candidate 1-itemset(C_1)

Item	TIDs	SC
a	{3, 4, 5, 7, 8, 9}	6
b	{1, 3, 4, 5, 6}	5
c	{0, 2, 3, 4, 5, 7, 8, 9}	8
d	{0, 3, 4, 5, 7, 8, 9}	7
e	{0, 1, 2, 3, 4, 6, 7, 8, 9}	9
f	{1, 3, 5, 6, 8, 9}	6
g	{0, 1, 3}	3
h	{0, 1, 5, 6, 9}	5
i	{0, 1, 3, 6, 8, 9}	6
k	{0, 7}	2
m	{0, 1, 2, 6, 7, 8, 9}	7
p	{0, 1, 3, 4, 5, 6, 7, 8, 9}	9

From the table above, the items viz., b, g, h and k are removed as infrequent because the items do not satisfied δ. The frequent 1-itemset is shown in Table 2. Since the common transactions (CTL) are stored in Table 3, they are removed from each item in L_1, the final L_1 is formed, and it is shown in Table 4.

Table 2. Frequent 1-itemset(L_1)

1- Itemset	TIDs								
a	3	4	5	7	8	9			
f	1	3	5	6	8	9			
i	0	1	3	6	8	9			
d	0	3	4	5	7	8	9		
m	0	1	2	6	7	8	9		
c	0	2	3	4	5	7	8	9	
e	0	1	2	3	4	6	7	8	9
p	0	1	3	4	5	6	7	8	9

Now the new_min is calculated by removing the number of items in CTL as $\delta_{new} = \delta - n = 6\text{-}2 = 4$. The logical buckets from final L_1, i.e. LB_1 and LB_2, are shown in Tables 5 and 6.

To reduce the storage space requirement further, this method finds the common transaction in which the all items occurs either by AND operation or intersection of the TIDs of all frequent 1-itemset. i.e.{3,4,5,7,8,9} ∩ {1,3,5,6,8,9} ∩ {0,1,3,6,8,9}∩{0,3,4,5,7,8,9}∩{0,1,2,6,7,8,9}∩{0,2,3,4,5,7, 8,9}∩{0,1,2,3,4,5,6,7,8,9}∩{0,1,3,4,5,6,7,8,9} = {8,9} and it is stored in CTL. The CTL is shown in Table 5.

Table 3. Common Transaction List(CTL)

CTL	
8	9

Table 4. Final Frequent 1-itemset(L_1)

1- Itemset	TIDs						
a	3	4	5	7			
f	1	3	5	6			
i	0	1	3	6			
d	0	3	4	5	7		
m	0	1	2	6	7		
c	0	2	3	4	5	7	
e	0	1	2	3	4	6	7
p	0	1	3	4	5	6	7

Table 5. Logical Bucket-1(LB_1)

1- Itemset	TIDs			
a	3	4	5	7
f	1	3	5	6
i	0	1	3	6

Table 6. Logical Bucket-2(LB_2)

1- Itemset	TIDs						
d	0	3	4	5	7		
m	0	1	2	6	7		
c	0	2	3	4	5	7	
e	0	1	2	3	4	6	7
p	0	1	3	4	5	6	7

The 2-itemset combinations viz., ad, am, ac, ae, fd, fm, fc, fe, id, im, ic, ie, dm, dc, de, mc, me, mp, and ce are in C_{2_1} and the items viz., ap, fp, ip, dp, mp, cp and ep are stored in C_{2_2}. The possible combinations viz., af, ai and fi need not be generated. It is shown in Tables 7 and 8 respectively.

Table 7. Candidate 2-itemset - Part I

C_{2_1}	TIDs	SC
ad	3, 4, 5, 7	4
am	7	1
ac	3, 4, 5, 7	4
ae	3,4,7	3
fd	3,5	2
fm	1,6	2
fc	3,5	2
fe	1,3,6	3
id	0,3	2
im	0,1,6	3
ic	0,3	2
ie	0, 1, 3, 6	4
dm	0	1
dc	0, 3, 4, 5, 7	5
de	0, 3, 4, 7	4
mc	0,2	2
me	0, 1, 2, 6, 7	5
ce	0, 2, 3, 4, 7	5

Table 8. Candidate 2-itemset - Part II

C_{2_2}	TIDs	SC
ap	3, 4, 5, 7	4
fp	1, 3, 5, 6	4
ip	0,1,3,6	4
dp	0, 3, 4, 5, 7	5
mp	0, 1, 6, 7	4
cp	0, 3, 4, 5, 7	5
ep	0, 1, 3, 4, 6, 7	6

The items viz., am, ae, fd, fm, fc, fe, id, im, ic, dm and mc are infrequent in C_{2_1} and no item is infrequent in C_{2_2}. Therefore, the frequent 2-itemsets are stored in L_{2_1} and L_{2_2} in jagged array notation as shown in Tables 9 and 10 respectively. The candidate 3-itemsets from L_{2_1} and LB_2 viz., adm, adc, ade, ace and dce, stored in C_{3_1} and the patterns adp, acp, iep, dep, mep, dcp and cep are kept in C_{3_2} as shown in Tables 11 and 12 respectively. The L_{3_1} and L_{3_2} are shown in Tables 13 and 14, respectively. Similarly, C_{4_1} and C_{4_2} are shown in Tables 15 and 16, respectively. L_{4_1} and L_{4_2} are $L_{4\_1} = \{\}$ and L_{4_2} is shown in Table 17.

Table 9. Frequent 2-itemset - Part I

L_{2_1}	TIDs				
ad	3	4	5	7	
ac	3	4	5	7	
ie	0	1	3	6	
dc	0	3	4	5	7
de	0	3	4	7	
me	0	1	2	6	7
ce	0	2	3	4	7

Table 10. Frequent 2-itemset - Part II

L_{2_2}	TIDs					
ap	3	4	5	7		
fp	1	3	5	6		
ip	0	1	3	6		
dp	0	3	4	5	7	
mp	0	1	6	7		
cp	0	3	4	5	7	
ep	0	1	3	4	6	7

Table 11. Candidate 3-itemset - Part I

C_{3_1}	TIDs	SC
adm	7	1
adc	3, 4, 5, 7	4
ade	3,4,7	3
ace	3,4,7	3
dce	0, 3, 4, 7	4

Table 12. Candidate 3-itemset - Part II

C_{3_2}	TIDs	SC
adp	3, 4, 5, 7	4
acp	3,4,5,7	4
iep	0, 1, 3, 6	4
dep	0, 3, 4, 7	4
mep	0, 1, 6, 7	4
dcp	0, 3, 4, 5, 7	5
cep	0, 3, 4, 7	4

Table 13. Frequent 3-itemset - Part I

L_{3_1}	TIDs			
adc	3	4	5	7
dce	0	3	4	7

Table 14. Frequent 3-itemset - Part II

$_3L_{\_2}$	TIDs				
adp	3	4	5	7	
acp	3	4	5	7	
iep	0	1	3	6	
dcp	0	3	4	5	7
dep	0	3	4	7	
mep	0	1	6	7	
cep	0	3	4	7	

Table 15. Candidate 4-itemset - Part I

C_{4_1}	TIDs	SC
adce	3,4,7	3

Table 16. Candidate 4-itemset - Part II

C_{4_2}	TIDs	SC
adcp	3, 4, 5, 7	4

Table 17. Frequent 4-itemset - Part II

L_{4_2}	TIDs			
adcp	3	4	5	7

Now, L_{4_1} is an empty list, so the algorithm terminates. It is observed from the experiment that the time needed for finding frequent items for sample dataset D in the example without the use of GPU is 0.8111 sec, whereas the wall time is 0.0073ms with GPU. The total memory requirement for the frequent itemset for the above dataset using the method in [18] is TM = 124+210+137+32=503 bytes. By using GNVDF, the memory requirement for the common transaction is CM = 2+2 = 4 bytes and the amount of memory saved using the proposed method is MS = (8×4) + {(7×4 + 7×4) + (2×4 + 7×4) + (0×4 + 2×4)} = 32 + 56 + 36 + 8 = 132 bytes. Therefore, the final memory requirement is TM_{final} = 503 - 132 = 371 which is 26.24% of memory saved for this example dataset compared to the memory requirement in [18]. It is also noted that the number of common transactions is directly proportional to the amount of memory saved.

4. Experimental Results and Discussion

The proposed algorithm was implemented using Python with CUDA Toolkit with NVIDIA GPU. An extensive experiment was conducted using four real-time datasets viz., chess, mushroom, t25i10d10k and c20d10k to evaluate the performance of GNVDF. The datasets and their details were shown in Table 18. They were obtained from the FIMI repository and an open-source Data Mining Library. The reason for choosing those datasets is that many researchers used those bench-mark datasets in Frequent Itemset Mining (FIM) and Association Rule Mining(ARM) based research. The runtime performance of the proposed method without GPU acceleration was obtained for each dataset, with the minimum threshold values ranging from 20% to 70% and is shown in Table 19. Similarly, the proposed algorithm was executed with GPU acceleration using the same minimum support range and results were tabulated in Table 20.

Table 18. Datasets used in experiments with their properties

Datasets	No. of transactions	No. of items	Average item count per transaction
chess	3196	75	37.00
mushrooms	8416	119	23.00
t25i10d10k	9976	929	24.77
c20d10k	10000	192	20.00

Table 19. Runtime (in ms) performance of the proposed algorithm without GPU

DS# → / MS* ↓	chess	mushroom	t25i10d10k	c20d10k
20	10759.6	14501.6	16332.5	16334.2
30	9845.5	13464.2	16225.8	16006.2
40	7972	11103.8	13885.7	15441.2
50	7101.7	10224.4	12645.6	14956.2
60	6293.4	9834	11101.2	13412.4
70	5082.2	8253	9256.4	12035.1

Table 20. Runtime (in ms) performance of the proposed algorithm with GPU-acceleration

DS# → / MS* ↓	chess	mushroom	t25i10d10k	c20d10k
20	119.5511	145.0160	161.7079	161.7248
30	107.0163	138.0940	156.0173	158.4772
40	83.9158	117.2770	129.7729	131.9761
50	73.2134	104.5091	108.3670	110.6496
60	64.2184	88.8096	102.4380	105.3511
70	53.4968	74.0512	83.6424	92.9924

#DS-Dataset *MS-min_sup(δ)

The graphical representation of the runtime performance of each dataset with and without GPU usage was illustrated in Fig.5. From tables 19 and 20, it was observed that when the number of items and transactions in a dataset increases, the time required for finding frequent patterns also increases. In general, there is an inverse relationship between the min_sup threshold and the time needed to determine the frequent patterns. i.e., when the min_sup threshold is increased, the number of generated candidate itemsets, followed by frequent patterns, is minimized, consuming less time for the higher threshold.

Fig.5. showed that the GPU acceleration significantly enables the execution speed of the proposed methodology, and GNVDF with GPU is faster by 90 to 135 times when compared with GNVDF without GPU acceleration. The reason for the performance enhancement is that the GPUs have many computing cores that allow the parallel execution of computation-intensive tasks. Since the GNVDF uses the VDF approach, the number of database scans is restricted to one [27] for determining each item's support count, which in turn reduces the overtime for finding the frequent patterns. But, VDF requires more memory for additional information like TID's than HDF [27], so a Jagged array has been used to minimise memory space is an advantage. Further, the elements in CTL removed from frequent 1-itemset save the memory space considerably more than the existing classical algorithms.

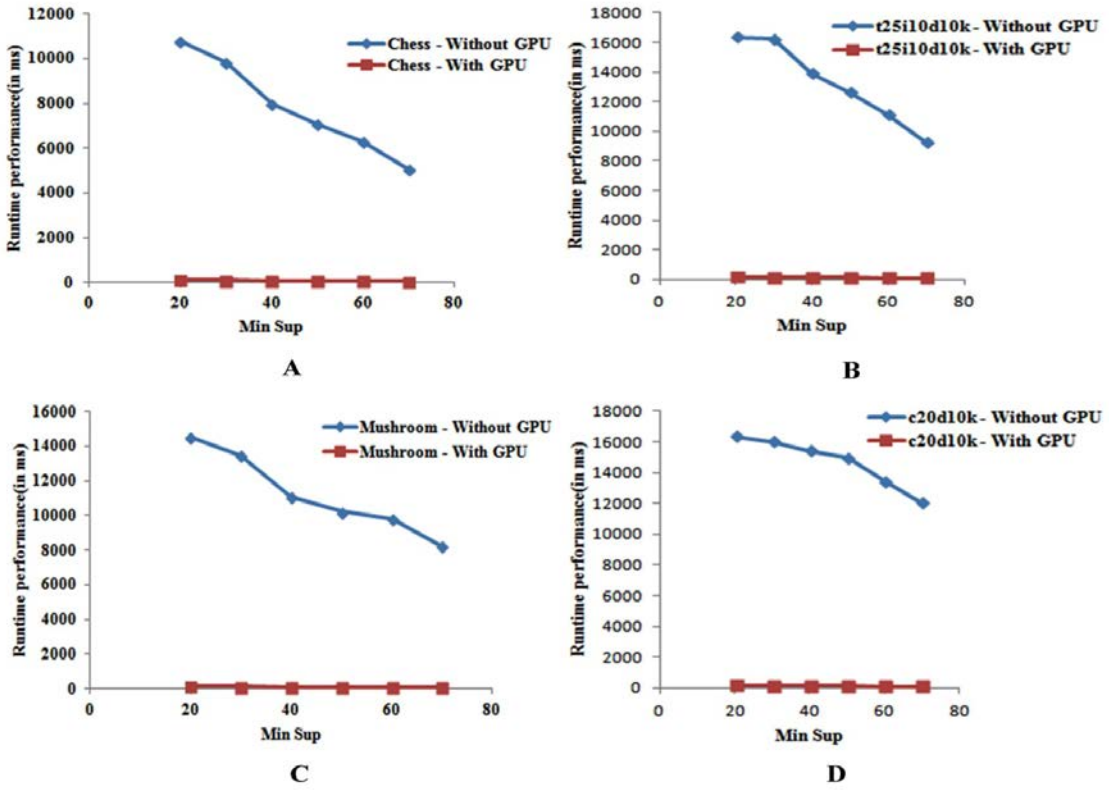

Fig. 5. Runtime performance of the proposed method with and without GPU acceleration of each dataset

5. Conclusion

A GPU-accelerated novel method for finding the frequent itemset called GNVDF has been proposed in this research article. It uses an innovative approach to discover the candidate and frequent itemsets by removing unnecessary itemsets to form the subsequent itemsets. It also utilizes GPU for speeding up the process. It also empowers the use of a jagged array storage structure and removes the common elements in 1-frequent itemsets. With GPU-acceleration and innovative way of determining itemsets, the time required is significantly decreased. Similarly, with a jagged storage structure, the memory requirement is also minimized than the classical algorithms. From the extensive experiments made, it is observed that the GNVDF with GPU is 90-135 times faster than with GNVDF without GPU and also proved that it suits both sparse and dense datasets. Further, the use of the VDF approach restricts the database scan to one.

References

[1] H. Hamidi and A. Daraei, "Analysis of Pre-processing and Post-processing Methods and Using Data Mining to Diagnose Heart Diseases," *International Journal of Engineering (IJE), TRANSACTIONS A: Basics*, vol. 29, no. 7, pp. 921-930, 2016.

[2] J. Han, J. Pei and M. Kamber, Data mining: concepts and techniques, Morgan Kaufmann Publishers, 2011.

[3] H. Lisnawati and A. Sinaga, "Data Mining with Associated Methods to Predict Consumer Purchasing Patterns", *International Journal of Modern Education and Computer Science(IJMECS)*, vol. 12, no. 5, pp. 16-28, 2020.

[4] A. Sinha, B. Sahoo, S.S.Rautaray and M. Pandey, "An Optimized Model for Breast Cancer Prediction Using Frequent Itemsets Mining", *International Journal of Information Engineering and Electronic Business(IJIEEB)*, vol.11, no.5, pp. 11-18, 2019.

[5] L. Vu and G. Alaghband, "A self-adaptive method for frequent pattern mining using a CPU-GPU hybrid model," *in Proceedings of the Symposium on High Performance Computing*, 2015.

[6] D. Albert, K. William, Fayaz and D. Veerabhadra Babu, "Exploiting Parallel Processing Power of GPU for High Speed Frequent Pattern Mining", *International Journal of Computer Engineering and Applications*, vol. 7, no. 2, pp. 71 - 81, 2014.

[7] W. Fang, M. Lu, X. Xiao, B. He and Q. Luo, "Frequent itemset mining on graphics processors," *in Proceedings of International Conference on Network and Parallel Computing*, 2009.

[8] S. M. Fakhrahmad and G. Dastghaibyfard, "An Efficient Frequent Pattern Mining Method and its Parallelization in Transactional Databases," *Journal of Information Science and Engineering*, vol. 27, no. 2, pp. 511-525, 2011.

[9] J. Zhou, K. M. Yu and B. C. Wu, "Parallel frequent patterns mining algorithm on GPU", *in Proceedings of International Conference on Systems*, 2010.

[10] D. William Albert, K. Fayaz and D. Veerabhadra Babu, "HSApriori: high speed association rule mining using apriori based algorithm for GPU," *International Journal of Multidisciplinary and Current Research*, vol. 2, pp. 759-763, 2014.

[11] M. Tiwary, A. K. Sahoo and R. Misra, "Efficient implementation of apriori algorithm on HDFS using GPU," *in Proceedings of International Conference on High Performance Computing and Applications*, 2014.

[12] J. Li, F. Sun, X. Hu and W. Wei, "A multi-GPU implementation of apriori algorithm for mining association rules in medical data," *ICIC Express Letters*, vol. 9, no. 5, pp. 1303-1310, 2015

[13] L. Vu and G. Alaghband, "A self-adaptive method for frequent pattern mining using a CPU-GPU hybrid model," *in Proceedings of the Symposium on High Performance Computing*, 2015.

[14] Y. Li, J. Xu, Y. H. Yuan and L. Chen, "A new closed frequent itemset mining algorithm based on GPU and improved vertical structure," *Concurrency and Computation Practice and Experience*, vol. 29, no. 06, pp. 1-12, 2016.

[15] K.W. Chon, S. H. Hwang and M. S. Kim, "GMiner: A fast gpu-based frequent itemset mining method for large-scale data," *Information Sciences*, vol. 439-440, pp.19-38, 2018.

[16] Y. Wang, T. Xu, S. Xue and Y. Shen, "D2P-Apriori: A deep parallel frequent itemset mining algorithm with dynamic queue," *in Proceedings of 10th International Conference on Advanced Computational Intelligence*, 2018.

[17] Y. Djenouri, D. Djenouri, A. Belhadi and A. Cano, "Exploiting GPU and cluster parallelism in single scan frequent itemset mining," *Information Sciences*, vol. 496, pp. 363-377, 2019.

[18] P. Sumathi, and S. Murugan, A Memory Efficient Implementation of Frequent Itemset Mining with Vertical Data Format Approach, International Journal of Computer Sciences and Engineering. 6(2018) 152-157.

[19] W. Gan, J. C. Lin, P. Fournier-Viger, H. C. Chao and P. S. Yu, "Survey of parallel sequential pattern mining," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 13, no. 3, pp. 1-34, 2019.

[20] Y. M. Guo and Z. J. Wang, "A vertical format algorithm for mining frequent item sets," *in Proceedings of 2nd International Conference on Advanced Computer Control*, 2010.

[21] E. Hashemzadeh and H. Hamidi, "Using a Data Mining Tool and FP-growth Algorithm Application for Extraction of the Rules in Two Different Dataset," *International Journal of Engineering (IJE), TRANSACTIONS C: Aspects*, vol. 29, no. 6, pp. 788-796, 2016.

[22] M. Samoliya and A. Tiwari, "On the Use of Rough Set Theory for Mining Periodic Frequent Patterns", *International Journal of Information Technology and Computer Science*, vol.8, no.7, pp.53-60, 2016.

[23] P. Prithiviraj and R. Porkodi, "A comparative analysis of association rule mining algorithms in data mining: a study," *American Journal of Computer Science and Engineering Survey*, vol. 3, pp. 98-119, 2015.

[24] F. Wang, J. Dong and B. Yuan, "Graph-based substructure pattern mining using cuda dynamic parallelism," *in Proceedings of International conference on intelligent data engineering and automated learning*, 2013.

[25] B. De Alwis, S. Malinga, K. Pradeeban, D. Weerasiri and S. Perera, "Horizontal format data mining with extended bitmaps," *in International Conference of Soft Computing and Pattern Recognition*,2011.

[26] P. Suresh, K. N. Nithya and K. Murugan, "Improved Generation of Frequent Item Sets using Apriori Algorithm," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 10, pp. 25-27, 2015.

[27] A.Subashini and M. Karthikeyan, "Itemset Mining using Horizontal and Vertical Data Format," *International Journal for Research in Engineering Application & Management*, vol. 05, no.03, pp. 534-539, 2019.

Authors' Profiles

P. Sumathi received her B.Sc and M.Sc degrees in Computer Science from Seethalakshmi Ramaswami College, affiliated to Bharathidasan University, Tiruchirappalli, India in 2001 and 2003 respectively. She received her M.Phil degree in Computer Science in 2008 from Bharathidasan University. She is presently working as an Assistant Professor in the Department of Computer Science, Vysya College, Salem. She is currently pursuing a Ph.D. degree in Computer Science at Bharathidasan University. Her research interests include Data Mining, Data structures and Database concepts.

Dr. S. Murugan received his M.Sc degree in Applied Mathematics from Anna University in 1984 and M.Phil degree in Computer Science from Regional Engineering College, Tiruchirappalli in 1994. He is an Associate Professor in the Department of Computer Science, Nehru Memorial College (Autonomous), affiliated to Bharathidasan University since 1986. He has 32 years of teaching experience in the field of Computer Science. He has completed his Ph.D. degree in Computer Science with a specialization in Data Mining from Bharathiyar University in 2015. His research interest includes Data and Web Mining. He has published many research articles in reputed National and International journals.

How to cite this paper: P. Sumathi, S.Murugan, "GNVDF: A GPU-accelerated Novel Algorithm for Finding Frequent Patterns Using Vertical Data Format Approach and Jagged Array", International Journal of Modern Education and Computer Science(IJMECS), Vol.13, No.4, pp. 28-41, 2021.DOI: 10.5815/ijmecs.2021.04.03