A Self-Adaptive Method for Frequent Pattern Mining using a CPU-GPU Hybrid Model

Lan Vu, Gita Alaghband
University of Colorado Denver
{lan.vu, gita.alaghband}@ucdenver.edu

ABSTRACT

Frequent pattern mining (FPM) is an important and computationally intensive task in data mining. We present a novel method, CGMM (CPU & GPU based Multi-strategy Mining), for mining frequent patterns that combines the computing power of CPU and GPU to speed up the frequent pattern mining. CGMM employs two different mining strategies and dynamically switches between them; the CPU-based strategy uses FP-tree data structure to perform the mining task on CPU while the GPU-based method converts the allocated data portions to bit vectors to work mainly on GPU. This unique approach has the following advantages compared to the existing methods: (1) utilizes the parallel processing capability of GPU for computationally intensive portions; the flexibility and low memory latency of CPU for the sophisticated data processing needed to manipulate the more complex data structures to enhance the overall performance (2) applies two mining strategies to efficiently mine both sparse and dense databases. The performance evaluation of CGMM on a machine with AMD CPUs and NVIDIA Tesla GPUs shows that in the best cases, the proposed method runs up to 229 times faster than well-known sequential FPM algorithms and 7.2 - 13.9 times faster than GPApriori, a GPU based algorithm for FPM. In addition to outperforming them, CGMM has more stable performance on both dense and sparse test datasets.

Author Keywords

Guides; instructions; author's kit; conference publication; keywords should be separated by a semi-colon.

INTRODUCTION

General-Purpose Graphics Processing Units (GPGPU) have emerged as powerful computing resources for general–purpose computing applications. They are used as co-processors capable of fast intensive computational processing that was once performed by CPUs [1, 2]. GPGPU applications are implemented using either CUDA [3] or OpenCL [4]. As the volume of data generated in most fields is fast growing, applying high performance techniques to enhance the overall performance of data analysis tasks has become important. Frequent pattern mining (FPM) is a crucial component of data mining used to find various types of relationships among variables in large databases such as associations [5], correlations [6], causality [7], sequential patterns [8], episodes [9] and partial periodicity [10] and has many practical applications such as market analysis, biomedical and computational biology, web mining, decision support, telecommunications alarm diagnosis and prediction, and network intrusion detection [11, 12].

HPC 2015, April 12 - 15, 2015, Alexandria, VA, USA

Motivation

Most existing FPM methods for GPU [13 - 22] are derived from Apriori [5], a sequential method that uses breadth-first strategy and the candidate generation-and-test approach to find frequent patterns as well as utilizes *downward closure* property (i.e. a k-itemset is frequent only if all of its sub-itemsets are frequent) to sharply reduce the search space. These features allow the GPU-based methods which create a large amount of workload with data suitable for presentation on GPU. However, for very large databases, the Apriori-like methods are significantly slower and consume much more memory in comparison to sequential methods like Eclat [23], FP-growth [24], and especially our proposed methods, FEM [25] and DFEM [26]. In many cases, Apriori is hundred times slower than FP-growth or its variants the FP-tree traversal time [27], H-mine [28], nonordfp [29], the use of FP-array data structure [30] and FP-growth with database partition projection [31]. Huang et al. [19] have shown that a parallel Apriori algorithm on GPU even run slower than sequential FP-growth run on CPU. Therefore, better mining strategy is essential. Additionally, Apriori performs efficiently on sparse data but not on dense databases [32, 33, 34], the GPU based Apriori methods for FPM exhibit the same poor performance on dense datasets as well.

Contribution

We propose CGMM for FPM which utilizes both CPU and GPU for its computation. The following features of CGMM contribute to its improved performance and distinguish it from the prior FPM methods:

1) CGMM consists of two different mining strategies, *CPUBasedMining* and *GPUBasedMining*, specifically designed to exploit the computing power of both CPU

and GPU. It uses a heuristic approach to dynamically select a suitable strategy for each data subset of the database based on its density characteristics during the execution.

2) The *CPUBasedMining* strategy uses only CPU to mine the frequent patterns by recursively constructing FP-trees without generating a large number of candidates. It is applied for data portions with sparse characteristics.

3) The *GPUBasedMining* strategy uses GPU as the main computing engine to mine the data portions with dense characteristics using a hybrid solution that consists of a new adaptive breadth-first approach, bit vector data structures, and candidate generation and test approach.

BACKGROUND

Problem Statement

The FPM problem is defined as follows: Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all distinct items in the transactional database *D*. The *support* of an *itemset* α, a set of items, is the percentage of transactions containing α in *D*. A *k-itemset* α, which consists of *k* items from *I*, is frequent if α*'s support* is larger or equal to *minsup*, where *minsup* is a user-specified minimum support threshold. Given a database *D* and a *minsup*, FPM searches for the complete set of frequent itemsets in *D*. For example, given the database in Table 1 and *minsup*=20%, the frequent 1-itemsets include *a, b, c, d* and *e* while *f* is infrequent because the *support* of *f* is only 11%. Similarly, *ab, ac, ad, ae, bc, bd, cd, ce, de* are frequent 2-itemsets and *abc, abd, ace, ade* are the frequent 3-itemsets.

Table 1. Sample dataset with *minsup* = 20%

Transaction ID	Items	Sorted Frequent Items
1	b,d,a	a,b,d
2	c,b,d	b,c,d
3	c,d,a,e	a,c,d,e
4	d,a,e	a,d,e
5	c,b,a	a,b,c
6	c,b,a	a,b,c
7	f	
8	b,d,a	a,b,d
9	c,b,a,e,f	a,b,c,e

Mining Frequent Pattern Using GPU

Mining frequent patterns is nontrivial because of its exponential search space, large amount of data and computational intensity. In the trend of applying high performance computing to increase the processing speed of data analysis tasks, developing FPM methods for GPU has received much interest because of the massive parallel capability of this device.

Modern GPUs have between dozens to hundreds of computing units/cores used as co-processors and can deliver much larger performance than a CPU for the right type of application. Their architectures typically consist of several streaming multiprocessors sharing same device memory. Each multiprocessor can have eight or more computing units/cores depending on the device model. The following general steps are needed for an application to run on a GPU: (1) copy input data from the main memory of CPU to the device (GPU) memory; (2) perform the computation on GPU; (3) copy the output data back from the device memory to the main memory. Two popular software platforms used to develop applications for GPU are CUDA [3] and OpenCL [4]. In CUDA computation is written as a *kernel* on the CPU to be launched on GPU with a massive amount of similar computational units known as *threads* [13]. A *kernel* launched to execute on GPU is referred to as a *grid*. A *grid* consists of multiple *thread blocks* where each block is a group of *thread*s. A *thread block* is assigned to a multiprocessor whose execution is in the form of SIMT (Single Instruction, Multiple Threads). The computing model of OpenCL is similar to CUDA. The GPU SIMT model of computation (which for most practical purposes is very similar to SIMD: Single Instruction, Multiple Data) works great for applications with massive amount of repetitive parallelism with regular access patterns. Performance penalties occur on GPUs due to irregular workloads, irregular access patterns, need for synchronization among threads belonging to different blocks, and need to move and access data in different levels of memory hierarchy.

The traditional FPM methods developed for sequential execution on CPU must be redesigned so that their computational model and data structure can adapt well to the GPU architecture. This is a highly challenging task for many of the complex applications designed for general purpose computing including FPM because GPU requires data presentation that can be processed uniformly and independently by a large number of concurrent *thread*s. In addition, data pre/post processing in CPU and transferring data between CPU and GPU can add a large enough overhead to the total execution time of FPM to negate the benefits of GPU.

Prior GPU-based FPM Algorithms

In this section, we review three most relevant GPU FPM algorithms CSFPM [17], GPApriori [20] and gpuDCI [18]

CSFPM (Candidate Slicing Frequent Pattern Mining) [17] is an Apriori-like method for GPU. It off-loads the most time consuming phase of counting to compute the supports to the GPU to speed up the total execution time. For better load balancing, the algorithm parallelizes and distributes the candidate itemsets to the GPU threads; each thread checks its own transaction in a candidate item. This reduces the processor waiting time since the load between processing units is more balanced.

GPApriori [20] is also an Apriori-like method for GPU. It maps the Apriori algorithm to the SIMD execution model by using bitset to represent the input database where the i^{th} bit in a bitset presents the occurrence of that item/itemset in the ith transaction of database (1: exist, 0: does not exist). This data structure improves upon the traditional approach of the vertical data layout in state-of-the-art Apriori implementations. Similar to CSFPM, GPApiori parallelizes only support counting step on the GPU while the remaining steps are executed on CPU. GPApriori applies several optimization techniques in its implementation: (1) before support counting is performed on GPU, candidates are preloaded to shared memory to prevent repeating global memory reads, (2) manual, hand-tuned loop unrolling to further improve the kernel speed; and (3) hand-tuned block size [20].

gpuDCI [18] is adapted from the DCI algorithm [35], a sequential mining approach that combines Apriori and Eclat. This algorithm starts its computation on CPU, as in DCI, and moves the pruned datasets to the GPU as soon as the bitwise vertical dataset fits into the GPU global memory. Afterwards the support computation is performed on GPU. However, after switching to GPU, the CPU still manages patterns, generates candidates and stores patterns that are frequent according to the supports computed by the GPU. Two parallel techniques were investigated: (1) for the transaction-wise technique, all GPU cores independent of the GPU multiprocessor they belong to, work on the same intersection or count operation; (2) for the candidate-wise technique, each GPU multiprocessor intersects and counts a different candidate. The candidate-wise technique has shown to perform better than the transaction-wise technique because it requires fewer synchronization operations.

NEW FREQUENT PATTERN MINING APPROACH USING A CPU-GPU HYBRID MODEL

Among many sequential frequent pattern mining methods that are traditionally developed for machines without GPU, FP-growth and its variants [27 – 31] are most efficient, especially for sparse large databases. They do not require generating a very large number of candidate itemsets as Apriori and Eclat do and hence save both memory and computation. The main mining computation of FP-growth is based on recursively generating FP-trees [24].

While the benefits of applying FP-growth cannot be ignored, developing a method based on FP-growth for FPM poses a lot of challenges for GPU due to FP-tree data structure, recursive tree construction, and tree traversal need. GPUs perform best for tasks whose data structures are linear and computations lend themselves well to vector processing. Moreover, it has been shown that FP-growth does not perform as well as Eclat when mining dense databases or mining with low minsups where the number of generated frequent pattern is very large [33, 34, 36]. For such cases, manipulating a very large number of FP-trees in FP-growth becomes more costly than intersecting the TID-lists of Eclat - the vertical layout of database where each list of a item/itemset stores IDs of transactions containing that item/itemset. It is important to keep in mind that for TID-lists, the vertical and linear data formats and list intersection operations of Eclat are quite suitable for GPU but the depth-first approach does not allow creation of enough parallel workload, compared to Apiori, to fully utilize the large computing resources of GPU.

Therefore, we combine and redesign the advanced features of FP-growth, Eclat, MAFIA [12] and Apriori into a new mining method, named CGMM, that applies both CPU and GPU computing to provide high FPM performance. As in DFEM, CGMM consists of two mining strategies and dynamically selects a suitable mining for each data portion of a database.

Overview of CGMM

CGMM consists of three main tasks: FP-tree construction, *CPUBasedMining* and *GPUBasedMining* described below. It starts with constructing the corresponding FP-tree followed by dynamically selecting between *CPUBasedMining* and *GPUBasedMining* similar to DFEM as depicted in Figure 1.

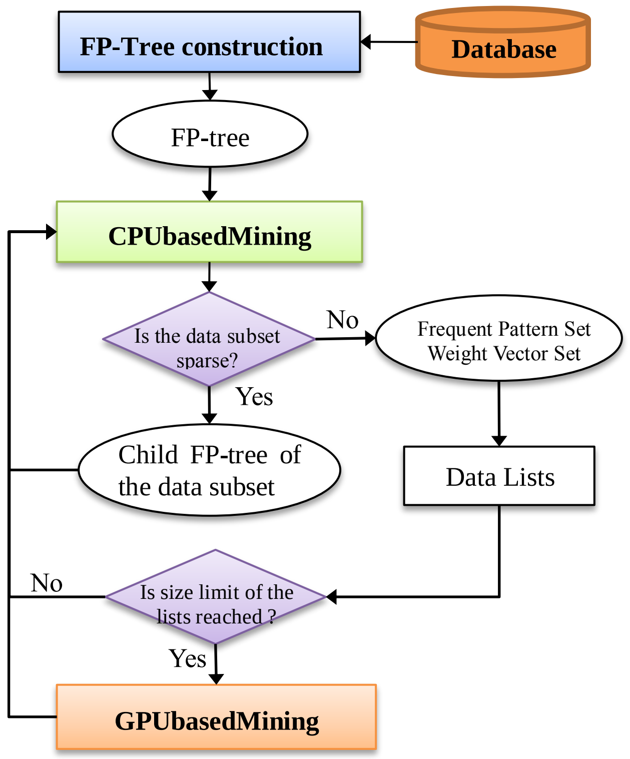
FP-Tree construction
Database
FP-tree
CPUbasedMining
Is the data subset sparse?
No
Frequent Pattern Set Weight Vector Set
Yes
Child FP-tree of the data subset
Data Lists
No
Is size limit of the lists reached ?
Yes
GPUbasedMining

Figure 1: The overview of CGMM

FP-tree construction (on CPU) reads the database to build the corresponding FP-tree. Use of this data structure significantly reduces the I/O cost because it compacts the database in memory before mining is performed. It also enables multiple mining strategies to be employed with relative ease as subsets of data from this tree can be used to create independent sub-mining tasks where each may use a different mining strategy.

CPUBasedMining (on CPU) extracts from the FP-tree the data subsets to recursively construct child FP-trees. The frequent patterns are identified based on newly generated FP-trees without the requirement of generating a large number of frequent pattern candidates. Because FP-tree data structure is complex and inefficient for mining on

GPU, only CPU is used to process the mining task. *CPUBasedMining* is distinct from the other mining methods because it is applied to sparse data subsets only. Determination of whether a data subset is sparse or dense is described in following section.

GPUBasedMining uses a new hybrid mining model designed for GPU applied to the dense data subsets. It presents data used to compute the *support* as bit vectors and maintains input and output data in data lists including *Frequent Pattern Set List* and *Weight Vector List*. The new frequent patterns are generated by applying candidate generation-and-test approach using a self-adaptive breath-first solution. Only a subset of frequent patterns is used to generate frequent pattern candidates at a time as long as their input and output data fit in the GPU memory. It addresses GPU memory limitation problem. Unlike the existing GPU solutions that off-load only the *support* counting phase to GPU, *GPUBasedMining* performs both the candidate generation and the *support* counting on GPU to increase GPU utilization and reduce overall processing on CPU as well as data transfer between CPU and GPU memories.

CGMM ALGORITHM

The CGMM algorithm (Figure 2) is performed in two stages. The first stage is loading data into memory by constructing the FP-tree. Then, frequent patterns are generated using the two mining strategies in our algorithms by initially invoking *CPUBasedMining*.

```
CGMM algorithm
Input: Transactional database D and minsup
Output: Complete set of frequent patterns
1:   Scan D once to identify all frequent items
2:   Scan D a second time to construct the FP-tree T
4:   Call CPUBasedMining(T,∅,minsup)
```

Figure 2: CGMM algorithm

FP-tree Construction

FP-tree is a prefix tree that compacts all sets of ordered frequent items from database into memory. This tree consists of a header table storing the frequent items with their *count*, a root node and a set of prefix sub-trees. Each node of the tree includes an *item name*, a *count* indicating the number of transactions that contain all items in the path from the root node to the current node, and a *link* to its parent node. Each linked list starting from the header table links all nodes of the same frequent item. If two itemsets share a common prefix, the shared part can be merged as long as the *count* properly reflects the frequency of each itemset in the database.

The construction of FP-tree requires two database scans. Only CPU is used for this stage because the tree data structure and operations are not suitable for GPU computing. Database is scanned the first time to find the frequent items and create the header table. A second database scan is done to get frequent items of each transaction. Next, these items are sorted and inserted in the FP-tree in frequency descending order. During the top-down traversal of the tree construction, if a node presenting an item exists, its count will be incremented by one. Otherwise, a new node is added to the FP-tree. Figure 3 illustrates an FP-tree constructed from the dataset in Table 1 where a pair <x:y> indicates *item name* and its *count*.

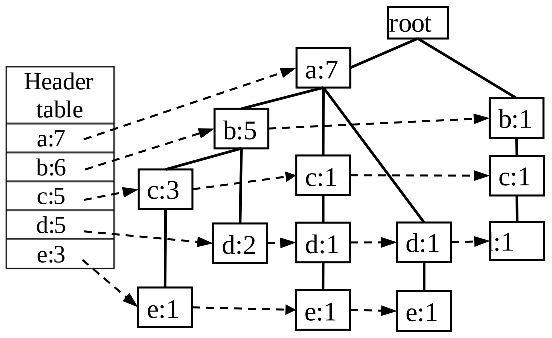

Figure 3: FP-tree constructed from the database in Table 1

CPUBasedMining

CPUBasedMining initializes the process of generating frequent patterns and mines the sparse data portions of the database. Similar to FP-tree construction stage, only CPU is used in this stage. The frequent patterns are reported by concatenating the suffix pattern of the previous step with each item α of the input FP-tree. At the beginning, this suffix pattern is Ø. Then, CGMM constructs a child FP-tree called conditional FP-tree for every item α using a data subset called conditional pattern base. This data subset is extracted from the input FP-tree of each recursive iteration and consists of sets of frequent items co-occurring with the suffix pattern. For example, the conditional pattern base of item d, which is extracted from the FP-tree (Figure 3) by bottom-up traversal starting from the nodes in the linked list of item d, consists of the 4 sets {a:2, b:2}, {a:1, c:1}, {a:1} and {b:1, c:1} in which {a , b} occurs twice (Figure 4-a). This base is used to construct the conditional FP-tree (Figure 4-b). The new tree is then used as the input of the next step of recursive iteration of this mining task.

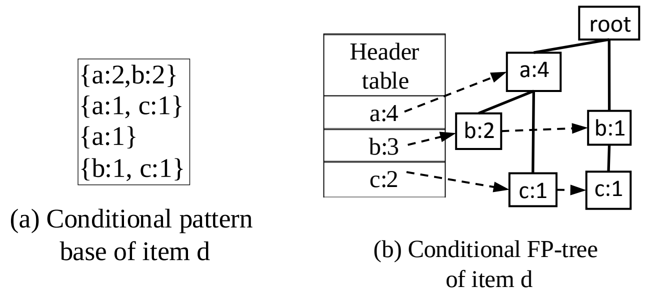

Figure 4: FP-tree constructed from the conditional pattern base of item d

CPUBasedMining does not process data subsets which have dense characteristics. Instead, it converts them into *Frequent Pattern Set* and *Weight Vector* and adds them into the *Frequent Pattern Set List* and *Weight Vector List* managed by GPUBasedMining. As the mining proceeds and when CPUBasedMining finds these lists full, it invokes GPUBasedMining to start the mining process on GPU. The algorithmic description of CPUBasedMining is in Figure 5.

```
CPUBasedMining algorithm
Input: FP-Tree T, suffix, minsup
Output: Set of frequent patterns
1:  If T contains a single path P
2:  Then For each combination x of the items in T
3:          Output $\beta$ = x $\cup$ suffix
4:  Else For each item $\alpha$ in the header table of T
5:  {   Output $\beta = \alpha \cup suffix$
6:      Construct $\alpha$'s conditional pattern base C
7:      size = the number of nodes in the linked list of $\alpha$
8:      If size > K
9:      Then { Construct $\alpha$'s conditional FP-tree T'
10:            Call CPUBasedMining (T',$\beta$,minsup)}
11:     Else { Create Frequent Pattern Set S and
12:                 Weight Vector W from C
13:            Add S into Frequent Pattern Set List $L_S$
14:            Add W to Weight Vector List $L_w$
15:            If ($L_S$ is full)
16:             Call GPUBasedMining($L_S$, $L_w$, minsup)
17:          }
18: }
```

Figure 5: CPUBasedMining algorithm

GPUBasedMining

GPUBasedMining mines the dense data portions of the database. It uses the GPU as co-processor for its most computational intensive need and CPU for the complicated tasks with data dependence to exploit the power and flexibility of GPU and CPU respectively.

Data Structures

GPUBasedMining uses several data structures to manage the mining data including *Frequent Pattern Set*, *Frequent Pattern Set List* and *Weight Vector List* (Figure 6).

Frequent Pattern Set is a set of frequent patterns that have same length *k* (i.e. they have *k* items in their itemsets) in which *(k–1)* items are common and one item is different among the frequent patterns in the set. For example, three frequent patterns abc, abd, abe can form a *Frequent Pattern Set because they have ab in common*. It contains a set of bit vectors where each presents the occurrence of a frequent pattern in the database. This data structure is used to generate new *(k+1)* length frequent pattern candidates and compute their *supports*. Some additional information of a *Frequent Pattern Set* includes *size* **-** the number of frequent patterns in the set, *length* **-** the number of items in a frequent pattern. Figure 6 demonstrates two *Frequent Pattern Sets* that are created from the conditional pattern bases of items e and c extracted from the FP-tree (Figure 3). A *Frequent Pattern Set* is only created if it satisfies the condition for GPUBasedMining and contains the data of at least two frequent patterns.

Frequent Pattern Set List works as a container that holds all *Frequent Pattern Sets* generated during the mining process and is updated by both CPUBasedMining and GPUBasedMining. In our example (Figure 6), two *Frequent Pattern Sets* are added into the *Frequent Pattern Set List.*

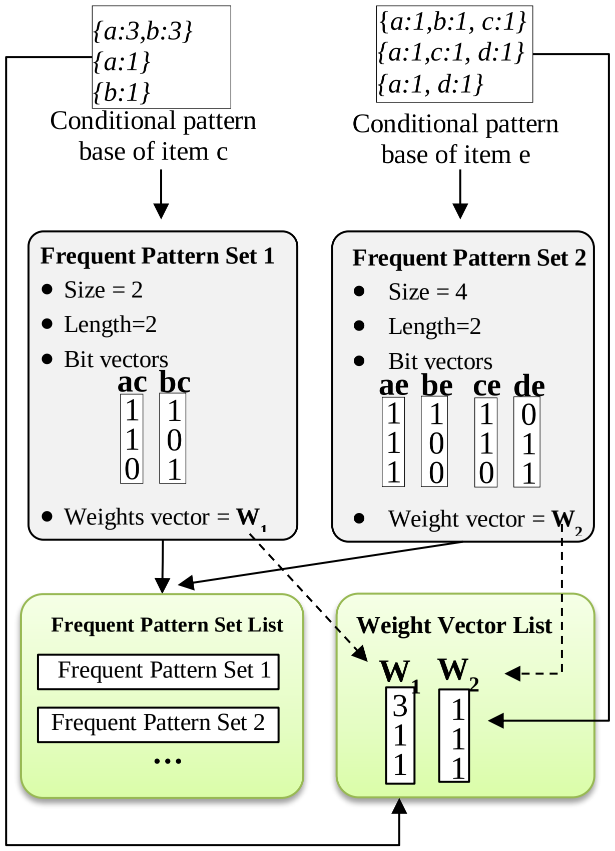

Figure 6: Data structures used by GPUBasedMining

Weight Vector List: the weight vector is a portion of *Frequent Pattern Set* that is used to compute the *support* of the patterns and is created by collecting the frequency values of sets in the conditional pattern base. Because many *Frequent Pattern Sets* that are newly generated by GPUBasedMining may share a same weight vector, we store this data in a separate list called *Weight Vector List* and add a reference in *Frequent Pattern Set* to its *weight vector* in the list to avoid duplication, save memory.

Algorithmic Descriptions

During the execution of CPUBasedMining, small data subsets that meet the condition to be mined using GPUBasedMining are converted to *Frequent Pattern Sets* and added into the *Frequent Pattern Set List*. When the number of items in this list reaches a predefined limit, GPUBasedMining is invoked by CPUBasedMining to start generating all new frequent patterns using *Frequent Pattern Set*s in the list. We present experiments showing the impact of different size limits on the performance of CGMM in next Section. The discovery of new frequent patterns from the *Frequent Pattern Set List* involves the following steps; note that Steps 2 and 3, which comprise the most computational intensive phases of the program, are being executed using GPU:

1. Extract a group of *Frequent Pattern Set*s from the list
2. Generate frequent pattern candidates using *Frequent Pattern Set*s
3. Compute the *support*s of candidates using *Frequent Pattern Set*s and *Weight Vector*s
4. Identify new frequent patterns from the candidates.
5. Add new *Frequent Pattern Sets* created from the new frequent patterns and repeat step 1 until no more *Frequent Pattern Set* is found from the list.

```
GPUBasedMining algorithm
Input: Frequent Pattern Set List $\mathbf{L_S}$,
       Weight Vector List $\mathbf{L_W}$, minsup
Output: Set of frequent patterns
1:  $\mathbf{M}$ = Available memory on GPU
2:  Copy $\mathbf{L_W}$ to the GPU
3:  While $\mathbf{L_S}$ is not empty
4:      Specify $\mathbf{S} \subset \mathbf{L_s}$ where $\mathbf{m} \leq \mathbf{M}$
5:          $\mathbf{m}$ = memory need of S and candidates of S
6:      Copy S to memory of GPU
7:      Generate candidates on GPU using S
8:      Compute support of candidates on GPU using S
9:      Copy support of candidates back to CPU
10:     FP = new frequent patterns with support $\geq$ minsup
11:     Output FP
12:     $\mathbf{S_{new}}$ = new Frequent Pattern Sets created using FP
13:     Consolidate bit vectors of FP on GPU using $\mathbf{S_{new}}$
14:     Remove S from $\mathbf{L_s}$
15:     Add $\mathbf{S_{new}}$ into $\mathbf{L_s}$
16: End While
```

Figure 7: GPUBasedMining algorithm

GPUBasedMining processes a group of *Frequent Pattern Set*s at a time by extracting multiple *Frequent Pattern Set*s from the list as long as their total memory size of newly generated candidates, their bit vectors and *Frequent Pattern Set*s used to generate them do not exceed the available memory on GPU. This workload computation helps CGMM flexibly scale to the memory size of physical device which is a major challenge in GPU computing. In addition, applying the data list structure allows GPUBasedMining working without recursion (recursive procedures do not generally yield high performance on GPUs). Figure 7 presents the algorithmic description of GPUBasedMining. In this figure, the computation steps involving the GPU include lines 1, 6 – 9 and 13 respectively and are detailed in the following section.

Generating Frequent Pattern Candidates and Computing Their Supports on GPU

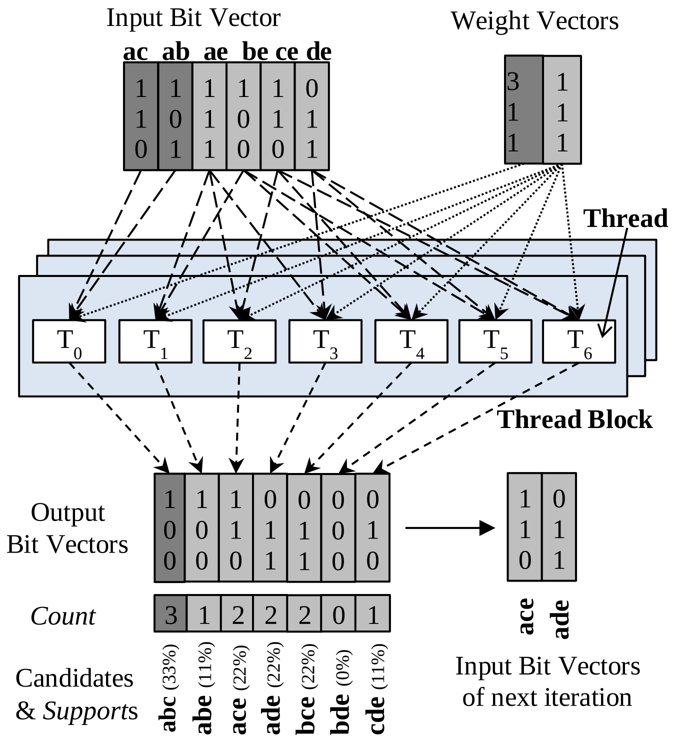

Figure 8: Generating pattern candidates and computing their *count*s on GPU

Unlike the related works that use GPU for the *support* counting phase only, GPUBasedMining performs both the frequent pattern candidate generation (step 2) and *support* counting (step 3) phases on GPU to reduce the computation handled by CPU; it better utilizes the GPU and therefore improves the efficiency of GPUBasedMining as well as the overall performance of CGMM. The *Frequent Pattern Sets* and the *Weight Vector*s are copied to GPU and distributed among the *thread block*s. Each concurrent thread in a *thread block,* based on the information of *Frequent Pattern Sets* assigned to its block, will identify the candidates it needs to work on and specify all the necessary information of these candidates such as the two parent frequent patterns, their input bit vectors, its output bit vector to store the result of ANDing the parent's bit vectors. Candidates are then generated in parallel by concurrent threads on GPU and their *support*s are computed. Each thread is responsible for the *support* of one or more candidates independently. Figure 8 illustrates the computations on GPU to generate the frequent pattern candidates and computation of the *support*s using the *Frequent Pattern Sets* and *Weight Vector*s in Figure 6.

Data Transfer Optimization: an important feature of GPUBasedMining is that the output bit vectors of bitwise operation on GPU are not copied back to the main memory. Instead, the bit vectors belonging to the new frequent patterns are consolidated and used as the inputs in the next iteration. For this reason, the new *Frequent Pattern Sets* ($\mathbf{S_{new}}$ in line 12 of Figure 7) which are stored in main memory do not include bit vector data. This technique minimizes the communication cost between CPU and GPU, saves memory on the CPU side and enhances the performance of CGMM. For example, for *minsup*=20%, the new frequent patterns are *abc, ace, ade, bce* because their *support*s > 20%. Among those, *abc, ade* are used to create a new *Frequent Pattern Set* to add to the *Frequent Pattern Set List* because they can be used to create the candidates *acde* in the next iteration. Therefore, the bit vectors of *abc, ade* are kept and unified in the memory of GPU.

PERFORMANCE EVALUATION

Experimental Setup

Datasets: They represent various characteristics and domains of interest for our experiments: three sparse, one moderate and two dense databases all obtained from the FIMI Repository [32], a well-known repository for FPM. The database features are reported in Table 2

Table 2: Experimental datasets of CGMM

Dataset	Type	# of Items	Average Length	# of Trans.
Chess	Dense	76	37	3196
Pumsb	Dense	2113	74	49046
Accidents	Moderate	468	33.8	340183
Retail	Sparse	16470	10.3	88126
Kosarak	Sparse	41271	8.1	990002
Webdocs	Sparse	52676657	177.2	1623346

Software: CGMM can be implemented using different programming platforms like CUDA [3] or OpenCL [4]. In our experiments, we choose CUDA to implement CGMM because CUDA delivers better performance for the Nvidia GPUs used in our experiments. We have carefully tested our implementation and have verified that it generates correct outputs in every case; this is often a challenge for complex applications developed on GPUs.

Hardware: We use an Altus 1702 machine with dual AMD Opteron 2427 processor, 2.2GHz, 24GB memory and 160 GB hard drive. This machine is equipped with NVIDIA Tesla Fermi C2050 GPU that has 3GB memory, 14 multiprocessors and each multiprocessor consists of 32 CUDA cores 1.5GHz. The operating system is CentOS 5.3, a Linux-based distribution.

Performance Evaluation

To evaluate performance of CGMM, we benchmarked it with six state-of-the-art FPM algorithms Apriori [5], Eclat [23], FP-growth [24], FP-growth* [27], AIM2 [33] and DFEM [26]. Unlike CGMM with multi-strategy approach using both CPU (sequential execution) and GPU (parallel execution), these algorithms apply only one mining strategy and use CPU as the only computing engine. The running time of the seven methods on six datasets with various *minsups* are given in Figure 9. The experimental results show that CGMM outperforms the other algorithms including Apriori, Eclat, FP-growth, FP-growth* and AIM2 on both dense and sparse datasets for most test cases. Please note that the y-axis of the graphs is in logarithmic scale. CGMM does not run as well as DFEM for larger *minsup* values but it outperforms DFEM when *minsup* reduces. Hence, we recommend to apply CGMM for applications that uses low *minsup* values. For test cases with low *minsup* values, CGMM runs 1.0 – 229 times faster five compared algorithms except DFEM on test datasets (Table 3). It runs faster than DFEM 1.0 – 1.8 times on Chess, Pumsb, Accidents, Kosarak and Webdocs datasets. For Retail, DFEM performs better than CGMM. However, their time difference reduces as *minsup* is set to smaller values. When *minsup* is set to smaller values, the number of data subsets mined by *GPUBasedMining* of CGMM is large and GPU is more efficiently utilized. When *minsup* is larger, the amount of work delegated to GPU is small and this device is under-utilized. In such cases, the highly optimized DFEM is a better FPM solution.

Table 3: Speedup of CGMM vs. other sequential algorithms

Datasets	Minsup	vs. Apriori	vs. Eclat	vs. FP-growth	vs. FP-growth*	vs. AIM2	vs. DFEM
Chess	20%	78.4	3.1	3.5	19.2	1.9	1.1
Pumsb	50%	229.2	2.0	2.5	13.3	5.7	1.3
Accidents	3%	N/A	1.3	1.6	6.8	7.2	1.0
Retail	0.003%	3.7	12.2	2.0	7.8	11.7	0.7
Kosarak	0.08%	15.8	14.6	6.7	12.3	1.0	1.8
Webdocs	4%	45.5	1.1	1.3	2.4	3.3	1.1

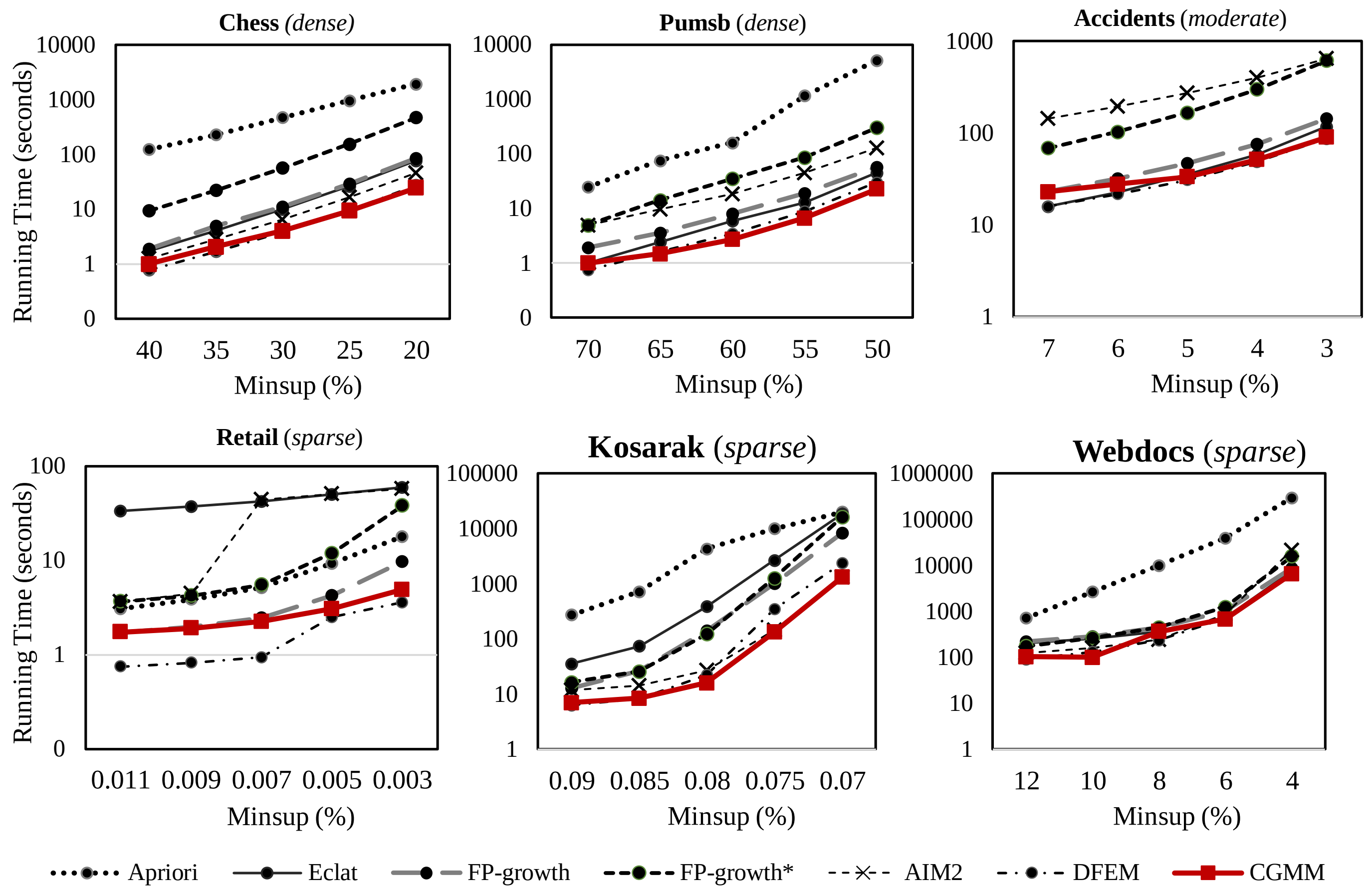

Figure 9: Running Time of CGMM vs. other sequential algorithms

We compare CGMM with GPApriori, a GPU based FPM algorithm [20]. Figure 10 shows CGMM runs 7.2-13.9 times faster than GPApriori on Retail dataset. In this test case, GPApriori uses GPU for entire dataset while CGMM uses GPU for only mining dense data subsets. GPApriori failed to run on other datasets because of the internal errors of this program.

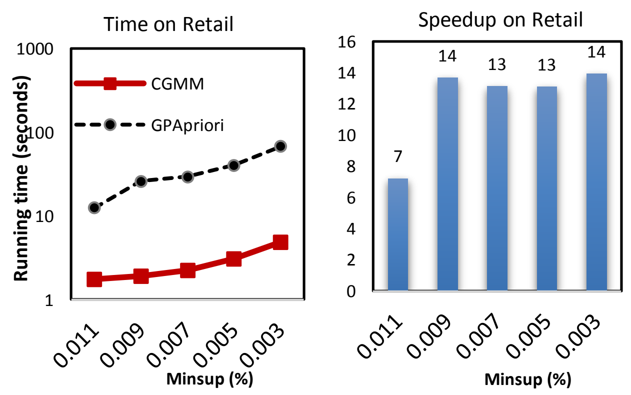

Figure 10: Time and speedup of CGMM vs. GPApriori

Upon the results on Retail, we find that for FPM problem, benefits of GPU is only obtained when we use it with suitable data structure and mining solution that can best leverage the computing power of GPU and adapt well to its limitation of memory and large data communication (e.g. mining dense data subset and low *minsup* values).

Impact of Using Multi-Strategy Approach in CGMM

To study the benefits of applying the two mining strategies, we measured the time of CGMM in three separate cases: (1) using CPUBasedMining only, (2) using GPUBasedMining only and (3) using the combination of CPUBasedMining and GPUBasedMining as intended. The experimental results in Table 4 show that combining the two mining strategies significantly reduces the running times for both sparse and dense databases compared to the cases where only one of the mining strategies were used. For example, CGMM with only CPUBasedMining took 1160 seconds to mine the Chess dataset while CGMM with both strategies ran in only 107 seconds which is 10.8 times faster. Similarly, for Accidents dataset, CGMM with both strategies performed 39.4 times faster than CGMM with only GPUBasedMining (i.e. 419 seconds vs. 16503 seconds). This performance gain comes from the ability to select the suitable strategy for each subset of data being mined to optimize the mining performance.

Additionally, we find that utilizing GPU for FPM does not always bring better performance for all types of data. Although the computing throughput of GPU is hundred times larger than one CPU, using GPU can sometimes downgrade the overall performance of a FPM task compared to using CPU because of GPU's memory limitations. It explains why for Accidents and Webdocs with very large memory requirements, running only GPUBasedMining performs much slower than running only CPUBasedMining. Combining the two and dynamic switching between them however results in improved overall performance.

Table 4: Runining time in seconds of CGMM using two mining strategy vs. using only one strategy

Dataset	CPUBasedMining	GPUBasedMining	CGMM
Chess	1160	180	107
Pumsb	2310	664	377
Accidents	547	16503	419
Retail	226	19	18
Kosarak	2773	841	680
Webdocs	6736	56017	6496

CONCLUSION

We have presented CGMM, a new CPU-GPU hybrid method for FPM. CGMM uses GPUBasedMining strategy for dense data subsets of database and CPUBasedMining for sparse ones. Our experimental results show that CGMM runs up to 229 times faster than six sequential algorithms on six real datasets and 7.2-13.9 times faster than GPApriori, a GPU based algorithm for FPM. Additionally, CGMM has the ability to self-balance the workload between its two mining strategies based on the characteristics of the database to execute significantly faster than using one mining strategy.

REFERENCES

1. T. Morgan. Top 500 supers the dawning of the GPUs. *http://www.theregister.co.uk/2010/05/31/top_500_supers_jun2010/*, May 2010.
2. Hou, R., Jiang, T., Zhang, L., Qi, P., Dong, J., Wang, H., Gu, X. and Zhang, S. Cost effective data center servers. In *Proc. 2013 IEEE 19th International Symposium on High Performance Computer Architecture* (2013), 179-187.
3. Nvidia. CUDA Programming. in *Best practices guide, http://www.nvidia.com/cuda*, 2013.
4. Munshi, A. OpenCL 1.0 Specification. in *Khronos OpenCL Working Group*, 2008.
5. Agrawal, R. and Srikant, R. Fast Algorithms For Mining Association Rules In Large Databases. In *Proc. 20th International Conference on Very Large Data Bases* (1994), 487-499.
6. Brin, S., Motwani, R. and Silverstein, C. Beyond Market Baskets: Generalizing Association Rules To Correlations. In *Proc. the 1997 ACM SIGMOD international conference on Management of data* (1997), 265-276.
7. Silverstein, C., Brin, S., Motwani, R. and Ullman, J. Scalable Techniques for Mining Causal Structures. *Data Mining and Knowledge Discovery* (2000), vol. 4, no. 2-3, 163-192.
8. Agrawal, R. and Srikant, R. Mining Sequential Patterns. In *Proc. the Eleventh International Conference on Data Engineering* (1995), 3-14.
9. Mannila, H., Toivonen, H. and Verkamo, A. Inkeri. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery* (1997), vol. 1, no. 3, 259-289.
10. Han, J., Dong, G. and Yin, Y. Efficient Mining of Partial Periodic Patterns in Time Series Database. In *Proc. the 15th International Conference on Data Engineering* (1999), 106-115.
11. Han, J., Cheng, H., Xin, D. and Yan, X. Frequent Pattern Mining: Current Status And Future Directions. *Data Mining and Knowledge Discovery* (2007), vol. 15, no. 1, 55-86.
12. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J. and Yiu, T. MAFIA: A Maximal Frequent Itemset Algorithm. *IEEE Transactions on Knowledge and Data Engineering* (2005), vol. 17, no. 11, 1490-1504.
13. Cui, Q. and Guo, X. Research on Parallel Association Rules Mining on GPU. In Proc. *the 2nd International Conference on Green Communications and Networks 2012* (2010), vol. 224, pp. 215-222, 2010.
14. Teodoro, G., Mariano, N., Jr.,W. M. and Ferreira, R. Tree Projection-Based Frequent Itemset Mining on Multicore CPUs and GPUs.in *Procceedings* of *the Symposium on Computer Architecture and High Performance Computing* (2010), 47-54.
15. Jian, L., Wang, C., Liu, Y., Liang, S., Yi, W. and Shi, Y. Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). *Supercomputing* (2013), vol. 64, 942-967.
16. Kozawa, Y., Amagasa, T. and Kitagawa, H. Parallel and Distributed Mining of Probabilistic Frequent Itemsets Using Multiple GPUs. *Lecture Notes in Computer Science, Database and Expert Systems Applications* (2013), vol. 8055, 145-152.
17. Lin, C.-Y., Yu, K.-M., Ouyang, W. and Zhou, J. An OpenCL Candidate Slicing Frequent Pattern Mining Algorithm on Graphic Processing Units. in *Procceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics* (2011), 2344- 2349.
18. Silvestri, C. and Orlando, S. gpuDCI: Exploiting GPUs in Frequent Itemset Mining. in *Procceedings of 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing* (2012), 416-225.
19. Huang, Y.-S., Yu, K.-M., Zhou, L.-W., Hsu, C.-H. and

Liu, S.-H. Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting. *Lecture Notes in Computer Science, Network and Parallel Computing* (2013), vol. 8147, 245-256.

20. Zhang, F., Zhang, Y. and Bakos, J. GPApriori: GPU-Accelerated Frequent Itemset Mining. in *Proc. the 2011 IEEE International Conference on Cluster Computing* (2011), 590-594.
21. Kozawa, Y., Amagasa, T. and Kitagawa, H. Fast Frequent Itemset Mining from Uncertain Databases using GPGPU. in *Proc. the Fifth International VLDB Workshop on Management of Uncertain Data* (2011), 892-901.
22. Zhou, J., Yu, K.-M. and Wu, B.-C. Parallel Frequent Patterns Mining Algorithm on GPU. in *Proc. the 2010 IEEE International Conference on Systems Man and Cybernetics* (2010), 435-440.
23. Zaki, M., Parthasarathy, S., Ogihara, M. and Li, W. New Algorithms for Fast Discovery of Association Rules. In *Proc. the 3rd International conference on Knowledge Discovery and Data Mining*, pp. 283-286, 1997.
24. Han, J., Pei, J. and Yin, Y. Mining Frequent Patterns Without Candidate Generation. In *Proc. the 2000 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp.1-12, 2000.
25. Vu, L. and Alaghband, G. A Fast Algorithm Combining FP-Tree and TID-List for Frequent Pattern Mining. In *Proc. the 2011 International Conference on Information and Knowledge Engineering* (2011), 472-477.
26. Vu, L. and Alaghband, G. Mining Frequent Patterns Based on Data Characteristics. In *Proc. the 2012 International Conference on Information and Knowledge Engineering* (2012), 369-375.
27. Grahne, G. and Zhu, J. Fast Algorithms for Frequent Itemset Mining Using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering* (2005), vol. 17, issue 10, 1347-1362.
28. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S. and Yang, D. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In *Proc. the IEEE International Conference on Data Mining* (2001), 441-448.
29. Racz, B. nonordfp: An FP-Growth Variation without Rebuilding the FP-Tree. In *Proc. the 2004 Workshop on Frequent Pattern Mining Implementations* (2004).
30. Liu, L., Li, E., Zhang, Y. and Tang, Z. Optimization of Frequent Itemset Mining on Multiple-Core Processor. In *Proc. the 33rd international conference on Very large data bases* (2007), 1275-1285.
31. Moriwal, R. FP-growth Tree for large and Dynamic Data Set and Improve Efficiency. *Information and Computing Science* (2014), vol. 9, no. 2, 559-583.
32. Frequent Itemset Mining Implementations Repository. *Workshop on Frequent Itemset Mining Implementation*, 2003-2004.
33. Shporer, S.. AIM2: Improved Implementation of AIM. In *Proc 2004 Workshop on Frequent Itemset Mining Implementations* (2004).
34. Schmidt-Thieme, L. Algorithmic Features of Eclat. In *Proc. 2004 Workshop on Frequent Itemset Mining Implementations* (2004).
35. Orlando, S., Lucchese, C., Palmerini, P., Perego, R. and Silvestri, F. kDCI: a Multi-Strategy Algorithm for Mining Frequent Sets. In *Proc. 2003 Workshop on Frequent Itemset Mining Implementations* (2003).
36. Fiat, A., and Shporer, S. AIM: Another Itemset Miner. In *Proc. 2003 Workshop on Frequent Itemset Mining Implementations* (2003).