Journal of Engineering and Applied Sciences 12 (12): 3098-3101, 2017
ISSN: 1816-949X

Parallelizing Pincer Search Using CUDA-A Conceptual Idea

Anchit Bhatia, Divyanshu Sharma and S. Chethan
Department of Information and Communication Technology,
Manipal Institute of Technology, Manipal, India

Abstract: Recent times have seen meteoric increase in the data that are available using which we can develop automated data-driven techniques of extracting useful knowledge. Data mining is the important step in this process of knowledge discovery. One of the key problem in most of the data mining applications is discovering the frequent item sets. Scanning of the huge data available to discover frequent item sets are computationally expensive. A conventional multi-core processor might not very effective multi-threading capabilities to be able to process large amounts of data leading to sequential implementation of a considerably large number of processes. Such sequential implementation leads to high computation times due to pipeline latency and other issues. Due to this limitation there is an increasing interest in the researchers to develop parallel data mining algorithms for faster implementation and efficient use of available GPU architectures. Pincer search is one the data mining algorithms which is used to discover the maximum frequent sets. Pincer search algorithm reduces both the number of times the database is scanned and also the number of candidate considered. In this study, we discuss a way to parallelize the pincer search algorithm to further speed up the process of discovering maximum frequent sets.

Key words: CUDA, data mining, pincer search, frequent sets, candidate

INTRODUCTION

Data mining is the process of discovery of valuable and non-obvious information from a usually big collection of data. Recent times have seen meteoric increase in the data that are available using which we can develop automated data-driven techniques of extracting useful knowledge. Data mining is the important step in this process of knowledge discovery. Given information and databases of sufficient size and quality, we can employ data mining techniques by providing capabilities of automated prediction of trends and behaviors and also automated discovery of previously unknown data to generate new business opportunities. As mentioned above, data mining algorithms help us in extracting some meaning out of pervasive data but comes with certain shortcomings.

Data mining techniques are flexible as in they can be implemented even if the existing platforms are upgraded. If we employ data mining tools by exploiting parallel processing systems then they can analyze extremely huge database within minutes. One of the key problem in most of the data mining applications is discovering the frequent item sets. One important reason for this is that it fits the "Market Based Model" and has wide application in real world scenarios. Such computation heavy tasks can be parallelized to save time and harness the powers of high performance GPGPUs. Scanning of the huge data available to discover frequent item sets are computationally expensive. A conventional multi-core processor might not very effective multi-threading capabilities to be able to process large amounts of data leading to sequential implementation of a considerably large number of processes. Such sequential implementation leads to high computation times due to pipeline latency and other issues. Due to this limitation there is an increasing interest in the researchers to develop parallel data mining algorithms for faster implementation and efficient use of available GPU architectures. Now that such many-core architectures are available we must harness the parallel computing paradigm.

Literature review: There have been previous endeavors to parallelize data mining algorithms. These aim at achieving the goal of utilizing the immense power of the available many core architectures provided by companies like NVIDIA and AMD. Here by we have enumerated other research papers which have used CUDA and NVIDIA for significant speedup in various algorithms. Adwa S. Al-Hamoudi and A. Ahmad Biyabani have succeeded implementing parallel code for KNN and decision tree with significant improvements in execution

Corresponding Author: Anchit Bhatia, Department of Information and Communication Technology,
Manipal Institute of Technology, Manipal, India

time. Roger Luis Uy and Nelson Marcos have worked on a very similar problem like our, dealing with frequent item sets (Al-Hamoudi and Biyabani, 2014). They succeeded in counting 1-item sets using SIMD architecture in CUDA. Similarly another very popular data mining algorithm has been optimized to run on GPUs by Al-Hamoudi and Biyabani (2014). They have succeeded in writing kernel code for partitioning around medoids originally proposed by Uy and Marcos (2016).

They have proved that with exponential increase in spatial data it is suitable to use GPU for PAM. In a similar domain Wang and Yuan (2014) proposed a methodology to build FP tree in a fully parallel implementation. Salah *et al*. (2015) have made use of parallel techniques to achieve fast mining of maximally informative k-item sets in big data and they have got a significant scale-up obtained with high item sets length and over very large databases.

Such previously published works show the immense potential of GPU based computing and how it has been proved that sequential algorithms can be optimized to run much faster on such architectures.

CUDA architecture: CUDA-an acronym for Compute Unified Device Architecture is an NVIDIA technology for general purpose GPU computing. This technology can harness the potential of the millions of CUDA-enabled devices even for non-graphics code.

In contrast to multi-core CPUs the many-core GPGPUs have the potential of performing thousands of operations in a parallel or simultaneously manner. The CUDA architecture comprises of compute engines for parallel processing combined with OS-kernel level support for initializing and configuring the hardware. Also there is a user-mode driver which has the provision of device level API for developers. Various libraries like BLAS and FFT are also optimized for CUDA.

Also wide variety of tools are available such as NVIDIA C Compiler (nvcc), CUDA Debugger (cudagdb), CUDA visual profiler (cudaprof). The ease of use with languages like Python is an added advantage. A GPU is visualized as a co-processor to the CPU having its own device memory. It runs many threads in parallel. The CUDA API for python helps us in writing both host and kernel code.

Each GPU constitutes of grids, blocks and threads. Each grid has multiple blocks and each block further has many threads. All threads in a grid execute the same kernel functions. The division of grid constitutes of 2D arrays of blocks. Each block is further organized as 3D array of threads. The dimensions are decided before the kernel is run. Figure 1 shows this hierarchy. Choice of

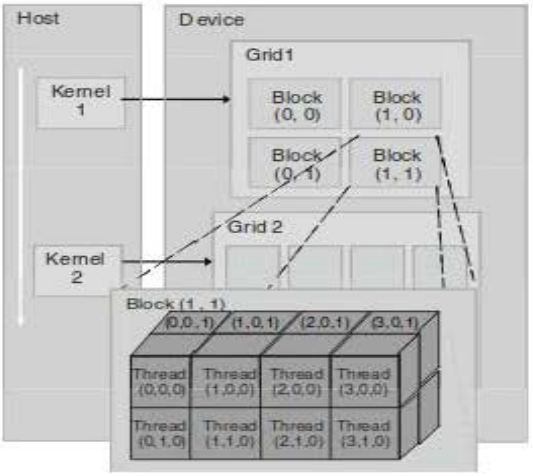

Fig. 1: Hierarchy of block dimensions and grid dimensions

block dimensions and grid dimensions is crucial to the kernel code and must be chosen wisely by the programmer. Threads belonging to same block also have properties that allow them to coorperate. The memory accesses by these threads is also to be managed and CUDA provides support for that too giving constructs like shared, constant and device memory to further reduce latency in data transfer.

Figure 2 shows how the concept of thread-ID and block-ID works. Each thread has its unique id in each block. We can have multiple unique ID's in each kernel code using different threads from different blocks. Parallel implementation implies that that very kernel code is executed by a grid and all threads from all the blocks run at the same time. Hence, the overhead of sequential iteration is avoided, allowing a much faster implementation.

Other than the thread hierarchy the GPU constitutes of streaming multiprocessors each of which constitutes of more processors known as streaming multiprocessors. Due to such an architecture the number of arithmetic operations a GPU can handle is much higher than a CPU. The important thing to be noted is the presence of each thread being mapped to different core, each core being able to run multiples threads at the same time. Hence, the latency due to one process in a thread does not delay other processes.

NVIDIA's parallel programming model is called SIMT-"Single Instruction, Multiple Threads". Two other different but related parallel programming models are SIMD-"Single Instruction, Multiple Data" and SMT-"Simultaneous Multithreading". Each model is based on a different source of parallelism.

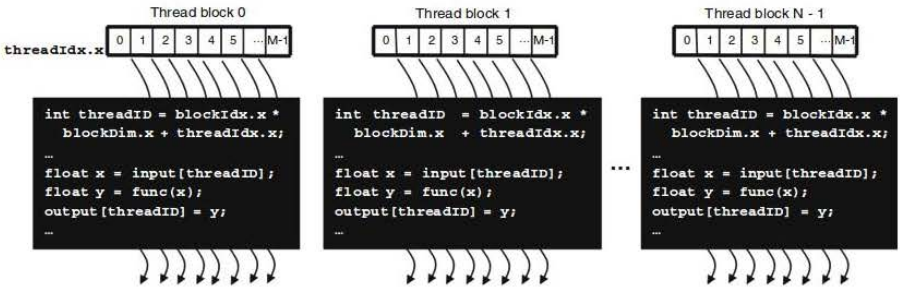

Fig. 2: The concept of thread-ID and block ID

Pincer search algorithm: Discovering frequent item sets is a key problem in important data mining applications which includes association rule mining (Lin and Kedem, 2002). Pincer search algorithm as shown in in Fig. 1 uses both top-down and bottom-up approaches to association rule mining. The main search is still the bottom-up where as a limited search is conducted using top-down approach which is used to update the Maximum Frequent Candidate Set (MFCS) a new data structure in pincer search algorithm. MFCS is actually a set which contains all the max frequent sets and is used to prune early candidates that would be normally encountered in the bottom-up search. The main advantage of the algorithm is that it does not require lucid review of every frequent itemset and it can deal with max frequent itemsets of large length. As discussed above, pincer search method combines bottom-up and top-down approaches. In bottom-up approach, first the subsets are generated and then the parent candidate set is generated using frequent subsets where as in top-down from the parent set subsets are generated. Suppose, say that a process in between an execution and some item sets have been already classified as either frequent or infrequent and few itemsets are yet to be classified. The approach is based on the perceptions that if an itemset is infrequent then all its supersets must be infrequent as well and if an itemset is frequent then all its subsets must be frequent (Lin and Kedem, 2002). The above two perceptions will speed up the process of searching max frequent set considerably.

Pincer search algorithm starts by generating 1-itemsets using top-down approach which prunes candidates in every pass which is done with the help of maximum frequent candidate set. MFS denotes Maximal Frequent sets which contains all the max frequent sets found during the execution which makes MFCS superset of MFS. Algorithm terminates when MFCS is equal to MFS. The pincer search algorithm is given (Lin and Kedem, 2002).

Algorithm 1:

Pincer search method

1. $L_o = \varnothing$; k = 1; $C_1 = \{\{i\} | i \in I\}$; $S_o = \varnothing$
2. MFCS = {{1,2, ..., n}}; MFS = ∅;
3. Do until $C_k = \varnothing$ and $S_{k-1} = \varnothing$
   a. Read the database and count support for C_k and MFCS.
4. MFS = MFS∪ {frequent itemsets in MFCS};
5. S_k = {infrequents itemsets in C_k};
6. Call MFCS_gen algorithm if $Sk \neq \varnothing$;
7. Call MFCS_pruning procedure;
8. Generate condidates C_{k+1} from C_k;
9. If any frequent itemset in C_k is removed from MFS_pruning procedure
10. Call recovery procedure to recover condidates to C_{k+1}.
11. Call MFCS_prune procedure to prune condidates in C_{k+1}.
12. k = k+1;
13. Return MFS

MFCS_gen

1. For all itemsets $s \in S_k$
   a. For all itemsets $m \in$ MFCS
   b. If s is a subset of m
   c. MFCS = MFCS\{m}
   d. For all items $e \in$ itemset a
      I. If m\{e}is not a subset of any itemset in MFCS
         ii. MFCS = MFCS∪{m\{e}}
2. Return MFCS

Recovery

1. For all itemsets $1 \in L_k$
   a. For all itemsets $m \in$MFS
      I. If the first k-1 items in 1 are also in m
      1. For i from j+1 to |m|
         2. $C_{k+1} = C_{k+1} \cup \{\{1.item_1, ..., 1.items_k, m.items_i\}\}$

MFS_prune

1. For all itemsets C in L_k
2. If c is a subset of any itemset in the current MFS
3. Delete c from L_k

MFCS_prune

1. For all itemsets in Ck_{+1}
2. If c is not a subset of any itemsset in the current MFC
3. Delete c from Ck_{+1}

MATERIALS AND METHODS

Pincer search uses the bottom up approach from Apriori algorithm and adds up its own method (top down approach) to reduce computation when frequent item set are very large. We propose to parallelize both of these constituents of the algorithm as follows:

Finding the support MFCS: In the algorithm 1, line 3a, deals with finding the support count for MFCS. This function can be parallelized by assigning each item in MFCS to one thread. The support of each item in the set of transactions is then calculated by each individual thread. This way we can achieve massive improvement in the computation speed by getting the support count of the MFCS set in only one transaction. Then, the total count can be added up. This is opposed to the normal serial implementation where we have to iterate amongst both MFCS and the transactions. This function is a part of the top down approach of pincer search.

RESULTS AND DISCUSSION

Finding the support candidate: In the algorithm 1, line 3a, deals with finding the support count for candidates. This function will be implemented in a similar way as finding the support MFCS to compute the support count of the items in the candidate set. The same approach is followed like the previous case and excessive iterations over the large dataset can be avoided. This is a constituent of the bottom-up approach.

Finding the frequent and non-frequent candidates: In the algorithm 1, line 5, deals with finding the l (frequent) and s(non-frequent) sets. In this function, each item in the candidate set is labelled either as frequent (l) or non-frequent(s) based on it's support count. To implement it in a parallel manner, each item is labelled by an individual thread to find the label of all items in one go. Finding the frequent and non-frequent candidates is a part of the bottom-up approach borrowed from the Apriori algorithm.

Finding the frequent mfcs: In the algorithm 1, line 4, deals with finding the frequent mfcs. This function is used to find out those items in mfcs that are frequent. To implement it parallely, each item is processed by an individual thread to find the whether it is frequent or not by matching it to the minimum support. Here again we deal with the top-down approach and propose to implement it in a parallelized way via, CUDA kernel code.

The library that could be used for the parallel implementation of the algorithm is numba cuda. In this library the CUDA kernels and device functions are compiled by decorating a python function with the jit or autojit decorators. Pincer search follows both bottom-up and top-down approaches. We can create CUDA kernel codes to parallelize functions that assist both the approaches. The various candidate counts and supports are calculated in the kernel code written using numba cuda.

CONCLUSION

In this research paper, we tried to discover all possible parallelizable functions of an important data mining algorithm: pincer search. For a real world scenario where the dataset are very large, improvement in computation times hold very high significance. Our proposal to parallelize the bottom up and some parts of top down counting approach of pincer search will make the algorithm even faster for scenarios where the datasets are massive and the frequent item set are also large. The assistance by parallel counting will show modest improvement in execution time.

REFERENCES

Al-Hamoudi, A.S. and A.A. Biyabani, 2014. Accelerating data mining with CUDA and OpenMP. Proceedings of the 11th International Conference on Computer Systems and Applications (AICCSA) 2014, November 10-13, 2014, IEEE, Riyadh, Saudi Arabia, pp: 528-535.

Lin, D.I. and Z.M. Kedem, 2002. Pincer-search: An efficient algorithm for discovering the maximum frequent set. IEEE. Trans. Knowl. Data Eng., 14: 553-566.

Salah, S., R. Akbarinia and F. Masseglia, 2015. Fast parallel mining of maximally informative K-itemsets in big data. Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM), November 14-17, 2015, IEEE, Montpellier, France, ISBN:978-1-4673-9504-5, pp: 359-368.

Uy, R.L. and N. Marcos, 2016. Fast 1-itemset frequency count using CUDA. Proceedings of the 2016 IEEE Conference on Region 10 (TENCON), November 22-25, 2016, IEEE, Manila, Philippines, ISBN:978-1-5090-2598-5, pp: 210-213.

Wang, F. and B. Yuan, 2014. Parallel frequent pattern mining without candidate generation on GPUs. Proceedings of the 2014 IEEE International Conference on Data Mining Workshop (ICDMW), December 14, 2014, IEEE, Shenzhen, China, ISBN:978-1-4799-4273-2, pp: 1046-1052.