Accelerating k-NN Classification Algorithm Using Graphics Processing Unit (GPU)

Selvaluxmiy Selvarajan, Velmakivan Ramakrishnan, Roshan Ragel, and Sampath Deegalla
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
luxmiy21@gmail.com, makivan8@gmail.com, roshanr@ce.pdn.ac.lk, dsdeegalla@pdn.ac.lk

Abstract — The data of government, enterprises, universities and various organizations are undergoing massive increases. Those data needs to be maintained. For this purpose, there are many algorithms used in data mining. Among them, k-Nearest Neighbor (k-NN) is mostly used in many applications, which are based on classification. When the data became significant, the execution time of the algorithm will be a bottleneck for data classification.

In this paper, we describe the implementation of the k-NN algorithm on GPU, using CUDA to improve the execution time of the k-NN algorithm for large datasets.

Keywords— Compute Unified Device Architecture (CUDA), Graphic Processing Unit (GPU), k-Nearest Neighbour (k-NN)

I. INTRODUCTION

Data mining is a process of collecting, searching through and analyzing a large amount of data in a database as to discover patterns or relationship. It is helpful in data cleaning, data pre-processing and integration of databases. It is used in many areas such as biological and market research, medical imaging, and other sectors. For example, in a criminal investigation, crime analysis includes exploring and detecting crimes and their relationships with criminals. Here text based crime reports can be converted into word processing files, and these are used to perform crime matching process.

There are a lot of algorithms for data mining, in particular for classification and clustering. Classification is a process of predicting the target classes for new objects called test dataset using training dataset, which is already classified. These algorithms are easy to implement, but they fall into trouble if the test set does not have any exact match in the training set. K- Nearest Neighbor algorithm (k-NN) is a solution for this problem. This algorithm selects a k number of training records, which are most close to a new object and then find the predominance of selected records.

The execution time is a significant barrier for the classification process when the dataset becomes large. Therefore, we need to accelerate the classification process. Acceleration process can be done by parallelizing the classification algorithm. For the parallelizing purpose, we use General Purpose Graphic Processing Unit (GPGPU), since it has thousands of cores and high memory bandwidth so that thousands of threads can be run simultaneously.

II. RELATED WORK

One of the research groups [1] used GPU implementation of Brute Force k-NN search to reduce the computation time. This was implemented using nVIDIA CUDA with two kernels. In one kernel, they compute the distance matrix size of m x n which contains the distance between the training set (size of m) and testing set (size of n). This distance computation is fully parallelized since data points are independent. In the other kernel, the distance is sorted for each test data concurrently. Optimized insertion sort is used for sorting the distance. It sorts the first k elements in the array using insertion sort. Then it checks the element (y) between k+1 and m. If the value of y is less than the k^{th} element of the sorted array, they insert that element to correct place in the sorted array. In our work, we implement the finding the nearest neighbor function using sequential search.

The researchers in [2] focused on accelerating email filtration using kNN on GPU. Emails contain different elements of formatting in addition to simple text. They used MailSystem.NET open source library to extract different components of the email including the number of recipients in the "TO", "CC" and "BCC" fields, validity of the address in the "TO" and "FROM" fields, the number of occurrence of the 50 most common spam keywords in the message and so on.

They have divided their implementation into three phases: a distance calculation phase, a sorting phase and a voting phase. In the distance calculation phase, each thread of the kernel calculating the distance between query point and training point and the results are stored in a struct in GPU memory. In the second phase, they divide the distance struct according to the test data. Then those arrays send to each block to find the smallest k distance of each test data. Finally, according to the values emails are classified into Spam or not. In our work, pre-processed datasets were used for classification. The datasets were read from the file and stored in arrays. And then sequential search method is used to finding the nearest neighbor phase. Therefore, the class array did not want to duplicate with each test record's distance array.

Another research group [3] has optimized the k-NN algorithm for exact similarity search on heterogeneous CPU – GPU systems by utilizing threshold compression with partial sort. Distance computation is done on a GPU and sorting is done on a CPU. In our work, distance calculation phase and finding the nearest neighbor phase were implemented in GPU

The above researches, serial and parallelized k-NN was implemented using same algorithms. In our research, K-NN algorithm was implemented using multiple algorithms on CPU and GPU. For example, the finding the nearest neighbor phase implemented using insertion sort, optimized insertion sort, bubble sort, and sequential search algorithms. And select the best algorithm which has minimal execution time. Therefore finding the nearest neighbor phase implemented using optimized insertion sort and sequential search on CPU and GPU respectively. And other implementations were same in both CPU and GPU.

III. BACKGROUND

A. k-NN Algorithm

k-NN is a well-known classification algorithm, which classifies a new object using the majority vote of neighbor's class. It is used in data mining applications, such as email spam filtering, content retrieval, gene expression, protein-protein interaction and 3D structure prediction and customer segmentation in online shopping websites since it gives minimal error rate and high accuracy.

The training dataset contains labeled dataset with multiple attribute values. Testing dataset is having undefined class value and multiple attribute values. And data can be text or numeric. The number of attributes should be the same for datasets.

This algorithm has three main steps.

1. Distance calculation.
   Distance can be calculated using Euclidian, Minkowski or Mahalanobis. Among them, Euclidian distance measure is used widely. The Euclidian distance is given by this equation (1),

$$D = \left(\sum (X_{(i)}-Y_{(i)})^{2}\right)^{1/2} \qquad (1)$$

   Here $X_{i,}$ Y_i are the attributes in testing dataset and training dataset respectively.

2. Finding the k- Nearest Neighbor
   In the second step, minimum k distance values will be sorted with their corresponding class values for the test data.

3. Voting
   In final phase locates the majority class value in the sorted training set and predicts that class to the test data.

B. CUDA and GPU

In early days, GPUs were mainly used for graphic processing. When nVIDIA is developing TESLA GPU architecture, they realized the programmers can use GPU like a processor and their process can be parallelizable. Therefore, nVIDIA added memory load and store instruction with additional bytes addressing to support a compiled C program.

GPUs have a high arithmetic intensity of operation and capacity to process parallel arithmetic operations. This is because it contains a larger number of ALU's than CPU and a smaller number of components for cache and flow control.

Cache memory is used to control the bandwidth requirements of applications so the data do not go back to DRAM when multiple threads that access the same memory. The bandwidth of GPU is nearly larger than 10 times or more of CPU.

In 2006, November nVIDIA introduced CUDA (Compute Undefined Device Architecture) with C/C++ Compiler, libraries and runtime software to work on GPU for parallel data computation model and develop applications.

CUDA is a parallel computing platform used for GPU. It helps to implement parallelism in the program without any knowledge on the low-level hard architecture of GPU. It is sometimes referred as Single Instruction Multiple Thread (SIMT) architectures. The Fig. 1 shows the thread organization in CUDA. There are three levels of thread abstraction:

1. Thread – Single process that executes an instantiation of a kernel.
2. Block – A 1, 2 or 3-dimensional collection of threads and they use shared memory and main memory for their process
3. Grid – 1 or 2-dimensional collection of blocks.

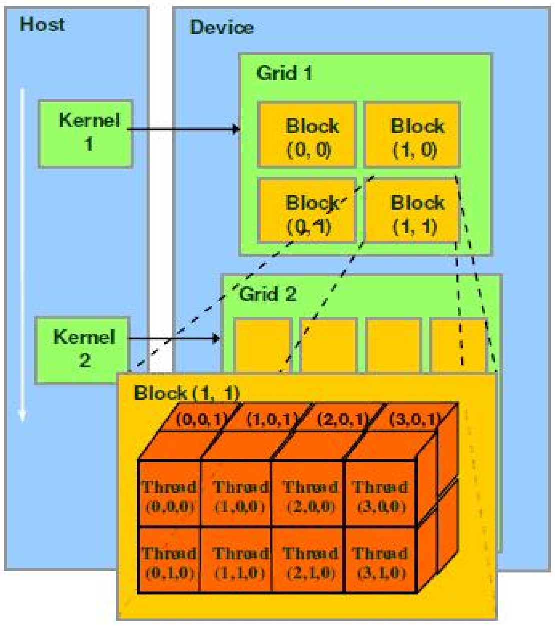

Fig.1. Thread Organization in CUDA

IV. k-NN Implementation in CUDA

Initially, we implement the basic serial version of the k-NN algorithm to run on CPU using C. k-NN classification execution time in CPU becomes large when the number of record and number of attributes is very large. Distance calculation and finding the nearest neighbor functions take more than 75% of the total execution time. Therefore to reduce the execution time we need to reduce the time consumption for distance calculation and finding the nearest neighbor functions. We are using GPU to parallelize those two functions.

In CUDA, we create two kernels, one for distance calculation and another one for finding the nearest neighbor. In distance calculation, we parallelize the distance calculation between one test data and all the training dataset. But here we cannot parallelize finding the nearest neighbor for one test data. But we can find the nearest neighbor for different test data set at a time. Using this method we reduce the time consumption for distance calculation and finding the nearest neighbor to accelerate the k-NN algorithm.

A. Distance calculation kernel

Here in each block, we send one test data, and all training data and each thread calculate the distance between that test data and a training data. We handle the block in 2D thread organization with block size 16x16. Fig. 2 shows the programming structure of the distance calculation CUDA kernel. This function returns distance in 1D array. Then this array will be passed into finding the k-nearest neighbor function.

```
1 __global__ void euclideanDistance() {
2
3 int row = blockIdx.x * blockDim.x + threadIdx.x;
4 int col = blockIdx.y * blockDim.y + threadIdx.y
5 distanceId = row * testRecords+col;
6
7 if (row less than trainRecords and col less than
                                         testRecords) {
8    for (i = 0 to attributes - 1) {
9       diff <= (d_trainSet [row*attributes + i] –
                              d_testSet[col*attributes + i])
10         sum += diff * diff;
11    }
12      distanceArray [distanceId] <= sqrt(sum)
13}
14}
```

Fig .2. Programming Structure of Distance Calculation kernel

In Fig. 2, line 9 and line 10 shows the Euclidean distance calculation method. The variable `distanceId` which is in line 5 illustrates the indexing of distances between a test and train data in the output distance array.

B. Finding the k-Nearest Neighbor Kernel

In this kernel, we define the number of thread is equal to the number of testing records in the set. Here we use sequential search method. Each thread finds the index of k minimum distance values of each test data and returns a 1D array of size k for each test record to a voting function, which is implemented in CPU. Fig. 3 shows the programming structure of sequential search kernel.

```
1 __global__ void NearestNeighbour(){
2
3 int threadId = blockIdx.x * blockDim.x + threadIdx.x;
4
5 if(threadId less than testRecords){
6    for(b = 0 to < k){
7      Min <= distanceArray[threadId]
8      index = 0;
9     for(c = 1 to c){
10       if(Min greater than distanceArray[c*testRecords
                                            + threadId]){
11         Min <= distanceArray[c*testRecords + threadId]
12         index <= c
13       }
14       if((Max less than distanceArray[c*testRecords +
                          threadId]) and (b equal to 0)){
15         Max <= distanceArray[c*testRecords + threadId]
16       }
17     }
18     distanceArray[index*testRecords + threadId] <=
                                                     Max
19     class[testRecords*b +threadId] <= trainClass[index]
20   }
21  }
22 }
```

Fig.3. Programming Structure for Finding the Nearest Neighbor

In Fig. 3, from line 6 to line 20 show the sequential search method. And line 19 show the how the index values of minimum distance stored in the final k-class output array with the size of k* number of test records.

V. Experiment Setup and Results

The CPU program is tested on Intel® Core ™ i5-3470 CPU @ 3.20GHz processor, and the CUDA program is tested on Tesla k40:2880 cores/30720 threads, 12GB memory.

We have considered the SHUTTLE numeric dataset from Machine Learning Repository website [5]. There are 43500 training data, 14500 testing data with nine attributes and missing values does not exist. It has 7 different class values. We store this dataset in CSV file format. We divide them into various sizes of training sets and test sets to evaluate the execution time of distance calculation phase, the k-nearest neighbor phase and the k-NN algorithm.

All the below results are taken for 43500 training record and for k=5. Table I shows the execution time of distance calculation phase of the k-NN algorithm on CPU and GPU.

TABLE I TOTAL ELEXCUTION TIME BETWEEN CPU AND GPU

Number of test records	**Execution time / sec**		**Speed Up (=T1/T2)**
	CPU (T1)	*GPU (T2)*	
500	1.42	0.44	3.23
2500	7.66	0.48	15.95
4500	14.45	0.53	27.26
6500	21.99	0.60	36.65
8500	30.57	0.60	50.95
10500	41.04	0.67	61.25
12500	51.50	0.74	69.59
14500	61.65	0.80	77.06

Table I shows the total execution time of the k-NN algorithm on CPU and GPU. Here we can notice that the total execution time on GPU is decreasing when compared to the total execution time on CPU. When the numbers of records are increased by 2000, the execution time on CPU and GPU also increase. But, the increasing rate of execution time on GPU is very small compared to the CPU's time increasing rate. As a result, the speed up rate is increased. It is the ratio between the execution time of CPU and the execution times on GPU. Fig. 4 shows the execution time versus the number of records in CPU and GPU. For 43500 training records and 14500 test records GPU is 77 times faster than CPU.

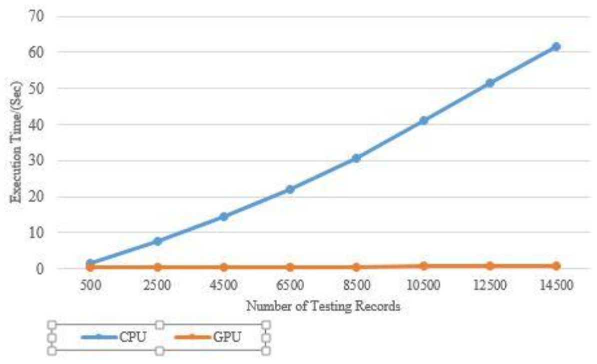

Fig.4 Execution Time VS Number of Records in CPU and GPU

VI. Conclusion

We presented a CUDA based k-NN classification algorithm implemented with two kernels, one for distances calculation and other for finding the nearest neighbor using search. The total execution time of the k-NN algorithm is reduced by 76 factors of CPU execution time for a standard dataset. The speed up factor is increasing the total size of the dataset. If the dataset is very large, then the speed up between GPU and CPU also becomes larger. The total execution time is reduced by using GPU for classification with the k-NN algorithm. Therefore, this method is more suitable for a large dataset with small dimensions.

Acknowledgment

We would like to thank the nVIDIA Research Center of University of Peradeniya who allowed us to use nVIDIA K40.

References

[1] Saravanan Thirumurukan, "A detailed introduction to k-nearest neighbor algorithm", 2010.

[2] Vincent Garcia, Eric Debreuve, Michel Barlaud, ''Fast k – nearest neighbor search using GPU'',Universit´e de Nice-Sophia Antipolis/CNRS Laboratoire I3S, 2000 route des Lucioles, 06903 Sophia Antipolis, France.

[3] Jiban K Pal, "Usefulness and application of data mining in extracting information from different perspectives", Library, Documentaion & Information Science Divition, Indian Statistical Institute, 2011.

[4] Wenbin Fang, Ka Keung Lau, Mian Lu, Xiangye Xiao, Chi Kit Lam, Philip Yang Yang, Bingsheng He, Qiong Luo, Pedro V. Sander, and Ke Yang, "Parallel data mining on graphics processors", Department of Computer Science and Engineering, Hong Kong University of Science and Technology Microsoft Research Asia, Microsoft China Co., Ltd.

[5] Statlog shuttle dataset, [Online]. Available at https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[6] Vincent Garcia, E´ric Debreuve, Frank Nielsen, Michel Barlaud, "k-nearset neighbor : fast gpu based implementations and application to high dimensional to high – feature matching", Laboratory dinformatique LIX, 91128 Palaiseau Cedex, France; Laboratories I3S, 2000 route des lucioles, BP 121, 06903 Sophia Antipolis Cedex, France; and Sony CSL 3-14-13 Higashi Gotanda, Shinagawa-Ku, 141-0022 Tokyo, Japan.

[7] Joshua M.Smithrud, Patric McElroy, Razvan Andony, "Massively parallel kNN using CUDA on spam – classification", Computer Science Department, Central Washington University, Ellensburg, WA, USA.

[8] Matsumoto and Man Lang Yiu, "Accelerating Exact Similarity Search on CPU – GPU SystemsTakazumi", Department of Computing, The hong Kong Polytechnic University, Hung Hom, Hong kong.

[9] D. Qiu, S. May, and A. N¨uchter, "GPU accelerated nearest neighbor search for 3D registration", in International Conference on Computer Vision Systems, Li`ege, Belgium, 2009.