Research on Parallelization of GPU-based K-Nearest Neighbor Algorithm

Hao Jiang and Yulin Wu

School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: hjiang@seu.edu.cn, daiboaidai@126.com

Abstract. Based on the analysis of the K-Nearest Neighbor Algorithm, the feasibility of parallelization is studied from the steps of the algorithm, the operation efficiency and the data structure of each step, and the part of parallel execution is determined. A K-Nearest Neighbor Algorithm parallelization scheme is designed and the parallel G-KNN algorithm is implemented in the CUDA environment. The experimental results show that the K-Nearest Neighbor Algorithm has a significant improvement in efficiency after parallelization, especially on large-scale data.

1. Introduction
In the background of the application of big data, the rapid excavation of useful information in massive data raises a higher demand for computational speed. GPU (Graphic Processing Unit) possesses strong computing power, and its own architecture can well meet the requirements of large-scale parallel computing[1]. Many researches on data mining algorithms have turned to parallel algorithms based on GPU. The K-Nearest Neighbor (KNN) algorithm is a basic classification and regression algorithm method proposed by Cover and Hart in 1968. Because of its good classification performance, KNN has been widely used in many fields. In this paper, the parallelization of KNN algorithm is studied. According to the CUDA programming characteristics and the GPU architecture used in this experiment, the parallel algorithm of KNN algorithm is designed. The tasks that can be parallelized are implemented on GPU, and the correctness of the parallelization algorithm and the improvement in execution efficiency are verified by experiments[2].

2. KNN Algorithm and Its Analysis
The main idea of the KNN algorithm is: Given a training data set, for a newly entered instance (where an instance can be regarded as a point in the feature space), If the majority of the k instances of the nearest neighbor to the instance belongs to a certain class in the training data, the input instance will be sorted to this classification[3]. Specific steps are as Algorithm 1.

The main problem with the implementation of the KNN algorithm is how to quickly search for k nearest neighbors, which is especially necessary when the dimensions of the feature space and the training data sets are large. The simplest method of solving this problem is linear scanning, or called the exhaustive method. It is divided into two steps. First, calculate the distance between the input instance and all instances in the training set based on the given distance metric; second, find the nearest K instances by sorting algorithm. Assuming that the size of the feature set of the training set is n, the training set capacity is N and the test set capacity is M, then the time complexity of the whole KNN algorithm is $O(M \times n \times N) + O(N \times M \times \log k)$[4].
The key part of KNN algorithm is distance calculation, especially when the size of feature space is large. How to optimize the distance calculation is the main problem to improve the KNN algorithm. In practical applications, the training samples are represented by feature vectors. To Euclidean distances, for example, the distance between vectors is actually equivalent to the multiplication of matrices [5].

Assuming that the capacity of the training set T is N, the capacity of the test set P is M, the feature space is the n-dimensional real vector space \( \mathbb{R}^n \), \( x_i, y_j \in \mathbb{R}^n \), \( x_i = (x_{i1}^{(1)}, x_{i2}^{(2)}, \ldots, x_{in}^{(n)}) \), \( y_j = (y_{j1}^{(1)}, y_{j2}^{(2)}, \ldots, y_{jn}^{(n)})^T \), then the distance between \( x_i \) and \( y_j \) is \( L(x_i, y_j) = (\sum_{l=1}^{n} |x_{il}^{(l)} - y_{jl}^{(l)}|^2)^{1/2} \). T and P can be expressed as follows in Eq. (1) and in Eq. (2), and defines L(T, P) in Eq. (3) as the matrix representation of the Euclidean distance between the instances in the test set P and the training set T; each column vector \( L_j = (L(x_1, y_j), L(x_2, y_j), \ldots, L(x_N, y_j))^T \) in L(T, P) is the vector representation of the Euclidean distance between the instance \( y_j \) in the test set P and the points in the training set T.

\[
T = \begin{pmatrix}
  x_{11} & x_{12}^{(2)} & \cdots & x_{1n}^{(n)} \\
  \vdots & \vdots & & \vdots \\
  x_{N1} & x_{N2}^{(2)} & \cdots & x_{Nn}^{(n)} \\
\end{pmatrix}_{N \times n}
\]

\[
P = \begin{pmatrix}
  y_{11} & y_{12}^{(2)} & \cdots & y_{1n}^{(n)} \\
  \vdots & \vdots & & \vdots \\
  y_{M1} & y_{M2}^{(2)} & \cdots & y_{Mn}^{(n)} \\
\end{pmatrix}_{n \times M}
\]

Algorithm 1: KNN Algorithm;
Input: Training data set \( T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \), wherein, \( x_i \in \mathbb{R}^n \) is the feature vector of the instances, \( y_i \in \mathbb{Y} = \{c_1, c_2, \ldots, c_k\} \) is the class of instances, \( i = 1, 2, \ldots, N \); the feature vector of the instances \( x \);
Output: Class \( y \) to which the instance \( x \) belongs.
Process:
(1) According to the given distance metric, the distance between the instance \( X \) and all data points in the training set \( T \) is computed as \( D \);
(2) In the \( D(x) \) sort, find the \( k \) points with the smallest distance, that is, the nearest \( k \) points, the neighborhood of the \( x \) that covers the \( k \) points is denoted as \( N_k(x) \);
(3) In the \( N_k(x) \) according to classification decision rules (such as the majority of votes) to determine the class \( y \) to which \( x \) belongs: \( y = \arg \max_{y_i} \sum_{x_i \in N_k(x)} I(y_i = c_i) \), \( i = 1, 2, \ldots, N; j = 1, 2, \ldots, k \); Where \( I \) is the indicating function, that is, when \( y_i = c_i \), \( I \) is 1, otherwise \( I \) is 0.
Each instance in the test set \( P \), \( L_i \) can be obtained by calculating the distance. Then sort the items in \( L_i \) to get the nearest \( k \) points to \( y_i \). Finally, the class of \( y_i \) is determined according to the classification rules (e.g., majority vote). The foregoing analysis shows that the efficiency of the \( k \) nearest neighbor algorithm is the time spent in calculating the distance between an instance and an \( n \)-dimensional training set and to sort the nearest \( k \) distances, especially the former.

3. Scheme Design of Parallel KNN Algorithm

Through the analysis of section II, in this paper, the parallel strategy of GPU-based KNN algorithm is divided into the following two parts.

3.1 Parallelization of distance calculations

The process of obtaining the distance matrix \( L(T, P) \) from the training set \( T \) and the test set \( P \) is similar to matrix multiplication. The difference is that each item of the result matrix in the matrix multiplication is the dot product of the row vector and the column vector, while each item in \( L(T, P) \) is the Euclidean distance computed by row vector and column vector. Therefore, the process of obtaining \( L(T, P) \) can be completely similar to the process of matrix multiplication, which can be accelerated in the CUDA platform through the GPU parallel computing and improve computing efficiency. In order to maximize the efficiency of matrix multiplication in the CUDA framework and make every thread in the GPU do more work, the matrices \( T \) and \( P \) can be divided into a number of small matrices. Each block is responsible for calculating the multiplication of the small matrix, making full use of the GPU's computing resources.

3.2 Parallelization using parity sorting algorithm

The idea of parity sorting is repeating scans in arrays twice. The first scan selects data items for each pair of a \([j]\) and a \([j + 1]\), where \( j \) is an odd number (\( j=1,3,5,…… \)). If the value of their keywords is reversed, then exchange them. The second scan performs the same operation on all even data items (\( j=2,4,6,…… \)). Repeat this both sorts until the array is all ordered. Because odd pairs and even pairs are independent of each other, each sort of order, odd pairs or even pairs can be compared and exchanged using the GPU multi-threaded, as to sort quickly. Specifically, assuming that the array size to be sorted is \( N \), the array is divided into \( N/(1024*2) \) one-dimensional blocks according to the size of the block. (In this experiment, the GPU thread block is assigned up to 1024 thread), making full use of the GPU's thread computing resources. Since every block and every thread within the block are parallel, so in the odds and even pairs of comparison and exchange in each trip, the comparison and exchange of each pair is done by a separate thread, and they are parallel.

To this end, this paper presents a GPU-based parallel algorithm, G-KNN algorithm, to improve the efficiency of the original KNN algorithm. The processes that CPU and GPU collaborate with are shown in Figure 1.

Specific steps are as follows:
1. Preprocessing the training set \( T \) and the test set \( P \) on the CPU and initialize \( k \);
2. Pass the required calculation data into GPU video memory;
3. The distance between matrix \( T \) and vector \( P \) is computed in parallel by GPU multi-core and multithreading;
4. \( L \) is the calculated distance matrix, and sort the column vectors in \( L \) corresponding to each test instance;
5. Returns the sorted distance matrix \( L \) back to CPU;
6. Select the \( k \) nearest neighbor distances of the test instance, and use the classification rules to
get the category of the test instance.

In order to maximize the efficiency of the distance calculation in the CUDA framework as well as to make every thread in GPU to do more work, the matrices T and P are divided into multiple block matrices. Each block is responsible for calculating the multiplication of the block matrix, and Make full use of the computing resources of GPU, as shown in Figure 2.[6]

As in Figure 2, using the shared memory of the GPU, the matrix data that needs to be calculated is read into the shared memory, and then the data in the shared memory will be calculated.

![Figure 1. The processing flow of G-KNN algorithm](image1.png)

![Figure 2 Matrix calculation model](image2.png)

This approach improves performance compared to reading data from a global memory every time. In the memory structure and thread mapping, the grid dimension is set to grid (M/WIDTH, N/HEIGH), the block dimension is set to block (WIDTH, HEIGHTH). For example, a block in L in the figure requires data surrounded by the T and P dashed lines, and each block has WIDTH*HEIGHT threads to complete the block calculation. The GPU thread block is allocated up to 1024 threads in the
experiments of this paper, thus in the specific implementation, WIDTH and HEIGTH are set to 32, so that GPU thread resources can be fully utilized.

4. Experiment and Analysis
The experimental platform is configured as: Intel(R) Core(TM) i7-4790K and the GTX980Ti of the NVIDIA Maxwell architecture. The experimental data set is a randomly generated N D-dimensional samples, and each sample has a label, which is identified based on the positive and negative of the first dimension data. 90% of the sample data was chosen as training data, and the remaining 10% was used as test data. As the experiment is mainly to verify the efficiency of G-KNN calculation, so the k in the comparison experiment takes 20. The distance measure adopts Euclidean distance, and the classification decision rules adopt majority voting method.

| Table 1 | The Time Comparison of Two KNN Algorithms When D = 20 |
|---------|------------------------------------------------------|
| N       | calculation time/s | G-KNN | KNN | speedup |
| 1000    | 0.62             | 2.11  | 3.40 |
| 2000    | 0.87             | 11.97 | 13.76 |
| 4000    | 1.36             | 64.28 | 45.79 |
| 8000    | 2.60             | 266.88| 102.65|
| 16000   | 6.94             | 1040.72| 149.96|

| Table 2 | The Time Comparison of Two KNN Algorithms When D = 40 |
|---------|------------------------------------------------------|
| N       | calculation time/s | G-KNN | KNN | speedup |
| 1000    | 0.65             | 2.25  | 3.46 |
| 2000    | 0.97             | 12.57 | 12.76 |
| 4000    | 1.48             | 67.54 | 45.6 |
| 8000    | 2.80             | 300.87| 107.45|
| 16000   | 8.55             | 1277.18| 149.37|

| Table 3 | The Time Comparison of Two KNN Algorithms When D = 60 |
|---------|------------------------------------------------------|
| N       | calculation time/s | G-KNN | KNN | speedup |
| 1000    | 0.66             | 2.32  | 3.51 |
| 2000    | 0.98             | 12.92 | 13.18 |
| 4000    | 1.62             | 78.07 | 48.19 |
| 8000    | 2.94             | 312.05| 106.14|
| 16000   | 9.64             | 1487.62| 154.31|

| Table 4 | The Time Comparison of Two KNN Algorithms When D = 80 |
|---------|------------------------------------------------------|
| N       | calculation time/s | G-KNN | KNN | speedup |
| 1000    | 0.66             | 2.49  | 3.77 |
| 2000    | 0.98             | 14.39 | 14.68 |
| 4000    | 1.77             | 82.40 | 46.55 |
| 8000    | 2.94             | 319.49| 108.67|
| 16000   | 11.51            | 1766.87| 153.50|
Table 1 to 4 respectively record the comparison results of the G-KNN and KNN algorithms when the data dimension D is from 20 to 80. Wherein the data set size N is from 1000 to 16,000. The calculation time and speedup ratio of the G-KNN algorithm and the KNN algorithm are shown in the tables.

Experimental result shows that, the distance calculation in the KNN algorithm occupies the entire algorithm for most of the time, and that especially as the data set and the data object dimension increase, distance calculation accounts for more than 90% of the time. Distance calculation in acceleration algorithm can improve the efficiency of the implementation of the algorithm. As it can be seen from the experimental results acceleration algorithm significantly improved in efficiency.

Figure 3 illustrates a comparison of the computation time between the KNN algorithm and the G-KNN algorithm when the data set size is N from 1000 to 16000 and the data dimension D=40. It can be seen that the advantages of the G-KNN algorithm in performance are not significant when the data set is small, but with the increasing size of the data set, the advantages of G-KNN algorithm are more and more obvious.

Figure 4 Speedup when D takes different value
Figure 4 illustrates the speedup ratio of the G-CPU algorithm relative to the KNN algorithm when the data set size is from 1000 to 16000 and the data dimension D is from 20 to 80. It can be seen that the speedup effect of G-KNN algorithm is obvious, and the speedup effect is not related to the value of D. With the increasing size of the data set, the speedup of the algorithm increases, and all exceed 140x when N=16000.

5. Summary
The G-KNN algorithm presented in this paper parallelized the distance calculation and sorting in the KNN algorithm. The performance of the algorithm has been improved significantly, especially in the calculation of the distance of the algorithm. The acceleration effect is more remarkable, and the acceleration ratio of the parallel algorithm increases with the increasing scale of the data set. The KNN algorithm is well suited for parallel execution on the GPU.

6. Acknowledgment
This work was supported by Collaborative Innovation Center of Wireless Communications Technology and the National Natural Science Foundation of China (Grant No. 61672152).

7. References
[1] WU En-Hua. State of the Art and Future Challenge on General Purpose Computation by Graphics Processing Unit[J]. Journal of Software. 2004(10), In Chinese.
[2] Garcia V, Debreuve E, Barlaud M. Fast k nearest neighbor search using GPU[C]. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2008: 1-6.
[3] Cover T, Hart P. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 1967, 13(1):21-27.
[4] Tian Pan, Hua Bei, Lu Li. Implementation of K-nearest Neigbor Algorithm Based on GPU[J]. Computer Engineering, 2015, 41(2): 189-192, 198, In Chinese.
[5] LI Hang. Machine Learning[M]. Tsinghua university press, 2012. 95-130, In Chinese.
[6] LI Xiao-Wen, CUI Xiang. Performance optimization of matrix multiplication and FFT in GPU. Modern Electronics Technique[J]. 2013, 36(4): 80-84, In Chinese.