Parallel Implementation of KNN Algorithm

Aparna R. Joshi
14CO204
Department of Computer Science and Engineering
National Institute of Technology Karnataka
Email: aparna29th@gmail.com

Isha Tarte
14CO217
Department of Computer Science and Engineering
National Institute of Technology Karnataka
Email: tarteisha@gmail.com

I. Problem Statement

K-nearest neighbor (KNN) is a widely used classification technique and has significant applications in various domains, especially in text classification. The computational-intensive nature of KNN requires a high performance implementation. In this project, we aim to produce a CUDA-based parallel implementation of KNN, using CUDA multi-thread model, where the data elements are processed in a data-parallel fashion.

II. Objectives

The aim of the project is to develop a GPU-based KNN (K Nearest Neighbours) algorithm. The KNN algorithm is a widely applied method for classification in machine learning and pattern recognition. KNN algorithm has a high computational complexity; the KNN algorithm consists of two phases - calculating distance between the query object and all the data points in the training dataset and finding k data points which are most similar. Similarity can be measured by the distance in the feature space, so this algorithm is called K Nearest Neighbor algorithm. After sorting the results of distances calculation, decision of the class label of the test point can be made according to the label of the k nearest points in the train data set. The distance between two points in the multidimensional feature space can be defined in many ways. Using Euclidean distance is usually the most ordinary and common method.

We aim to compare the performance of the KNN algorithm in CPU with that obtained with the GPU (CUDA implementation). We will also compare the performance of various sorting algorithms for KNN such as insertion sort and merge sort.

III. Project Timeline

- **Feb 21 - Feb 28** - Project topic selection, project proposal submission, project timeline and workload distribution
- **Mar 1 - Mar 7** - Understanding the working of KNN, selection/generation of datasets, serial implementation of KNN
- **Mar 8 - Mar 15** - Kernel implementation of Phase 1 (distance calculation) in CUDA and insertion sort for Phase 2 (for sorting and selection).
- **Mar 16 - Mar 23** - Exploring efficient sorting algorithms that can be parallelized and implementing them in CUDA for KNN. Comparing the performance of various sorting algorithms for KNN.
- **Mar 24 - Mar 31** - Extending KNN for regression. Here the output will be the mean of k nearest train data points. Comparing CPU and GPU performance of KNN.
- **Apr 1 - Apr 8** - Comparing the performance of GPU based KNN for data with different dimensions. Plotting the performance curves.
- **Apr 9 - Apr 12** - Code cleaning, increasing code readability, documentation. Final demo.

IV. Work Distribution

Aparna R. Joshi:

- Project topic selection
- Understanding the working of KNN,
- Selection of any available standard datasets (eg. UCI)
- Serial implementation of KNN
- Kernel implementation of Phase 1 (distance calculation) in CUDA using Euclidean distance
- Comparing the performance of GPU based KNN for data with different dimensions
- Plotting the performance curves.
- Code cleaning and report

Isha Tarte:

- Project topic selection
- Understanding the working of KNN
- Generating synthetic datasets
- Serial implementation of KNN
- Implementation of distance kernel using Manhattan distance
- Comparing the performance of GPU based KNN for data with different dimensions
- Plotting the performance curves.
- Documentation

V. Mid Progress

In summary, the project has been progressing as per our stated timeline.
List of objectives achieved:

- Understood the working of KNN.
- Serial implementation of KNN for a single query object (regression) where insertion sort was used for sorting.
- Serial implementation of KNN for a single query object (classification) where insertion sort was used for sorting.
- Parallel implementation of KNN for a single query object (regression).
- Comparison of execution time of serial and parallel implementation.

We understood the working of KNN, selected from the UCI machine learning repository a dataset for initial code testing - Wine Quality Data Set

- Data Set Characteristics: Multivariate
- Attribute Characteristics: Real
- Number of Instances: 4898
- Number of Attributes: 12

VI. About KNN algorithm

KNN algorithm is widely applied in pattern recognition and data mining for classification, which is famous for its simplicity and low error rate. The principle of the algorithm is that, if majority of the k most similar samples to a query point q_i in the feature space belong to a certain category, then a verdict can be made that the query point q_i fall in this category. Similarity can be measured by the distance in the feature space, so this algorithm is called K Nearest Neighbor algorithm. A train data set with accurate classification labels should be known at the beginning of the algorithm. Then for a query data q_i , whose label is not known and which is presented by a vector in the feature space, calculate the distances between it and every point in the train data set. After sorting the results of distances calculation, decision of the class label of the test point q_i can be made according to the label of the k nearest points in the train data set.

Each point in d-dimensional space can be expressed as a d-vector of coordinates, such as:

$$p = (p_1, p_2, ..., p_n) \tag{1}$$

The distance between two points in the multi-dimensional feature space can be defined in many ways. Using Euclidean distance is usually to be the most ordinary method, that is:

$$dist(p, q) = \sqrt{\sum_{i=1}^{i=n} (p_i - q_i)^2} \tag{2}$$

The quality of the train data set directly affects the classification results. At the same time, the choice of parameter K is also very important, for different K could result in different classification labels.

Serial implementation of KNN was completed as per the timeline proposed. In serial implementation, we used euclidean distance for calculating the distance between the query object and training set. The second phase includes sorting of the distances calculated to find the k nearest neighbours.

Parallel Implementation: The computation of the distances can be fully parallelized since the distances between pairs of tuples are independent. This property makes KNN perfectly suitable for a GPU parallel implementation. In our implementation, after transferring the data from CPU to GPU, each thread performs the distance calculation between the unknown/query object and a training object. The training set is loaded into the shared memory of each SM from the global memory. Each SM manages a unique portion the training set. Threads in a common block share the training objects with others. Thus, a large number of threads and blocks are launched. In this way, the distance calculation is parallelized in a data-parallel fashion.

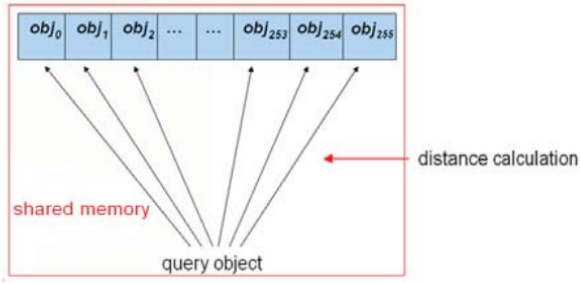

Fig. 1. Illustration of the distance calculation in parallel

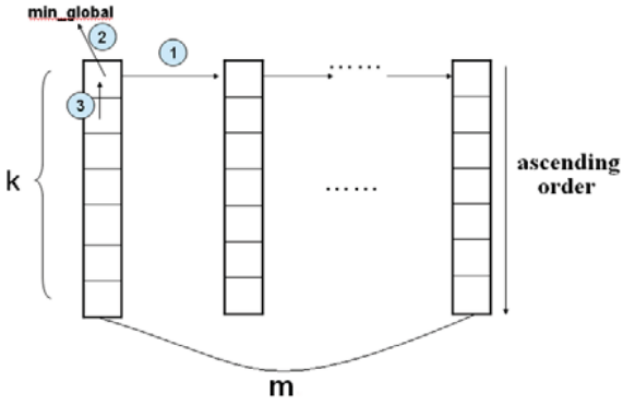

Fig. 2. Global-k nearest neighbors from m local-k nearest neighbors

After calculating the distances between the unknown object, p, and the training objects, sorting is performed to find the k nearest neighbors to p. First, the distances between p and the objects in a block are stored in the shared memory. Each thread takes care one distance. For a given thread, it obtains the rank of its distance by comparing it with other distances in

the shared memory. Such ranks are generated simultaneously so that the nearest k neighbors of p in the block are obtained, called local k nearest neighbors. Also, the local k neighbors are sorted according to the rankings. Second, we need to generate the global k nearest neighbors across all the blocks from the local k ones. How this is implemented is: the thread scans the smallest ones of each local k. Since the local k is a queue in ascending order, the first one in the queue is the local smallest. The smallest one of m local ones is selected and it's the global smallest. Then the second smallest one of the selected queue is compared with the rest (m-1) local smallest ones. In this way, each scan generates one nearest neighbor. With only k*m scans, the global k nearest neighbors of p are obtained.

The kernel implementation of phase 1 that is the distance calculation kernel as well as the phase 2 for sorting is complete. The kNN algorithm is performed for a single query or test object (this will be extended to multiple query objects - a different kernel) and for regression (can be extended by taking a majority vote for classification). Preliminary results of comparison between the serial and the parallel/GPU implementation are obtained and a graph that is shown in 5 is plotted. We can see that the GPU performs much better than the CPU. Next steps are to explore and implement more efficient (and specially suited for parallel programming) sorting algorithms on the GPU and compare with the serial version; also to test codes on more synthetic/acquired datasets with varying characteristics and see how performance changes with different dimensionalities of the datasets.

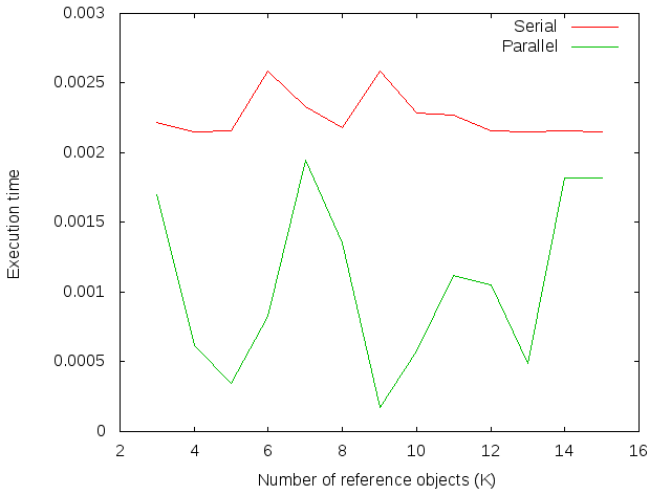

Fig. 3. Comparison of execution time

VII. Conclusion

We present a parallel implementation of KNN. The algorithm implemented is actually a hybrid implementation of the CPU and the GPU. The GPU performs two CUDA kernels: distance calculation and nearest-neighbors sorting. Data elements in these two kernels are processed in a data-parallel fashion. As the emergence of the CUDA programming model, GPU has become a promising platform for supercomputing.

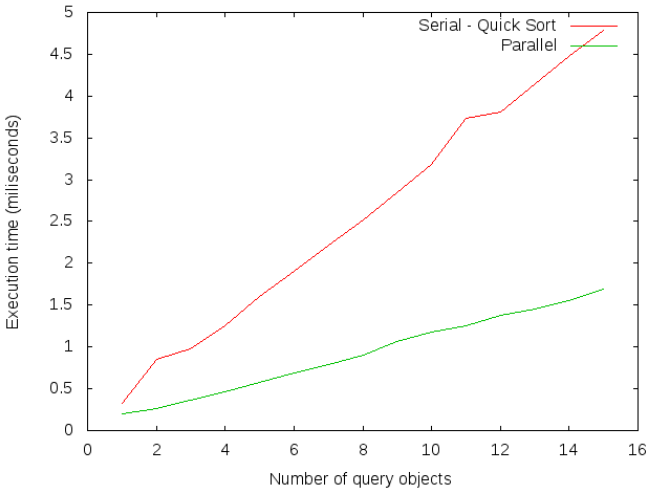

Fig. 4. Comparison of execution time wrt number of query objects

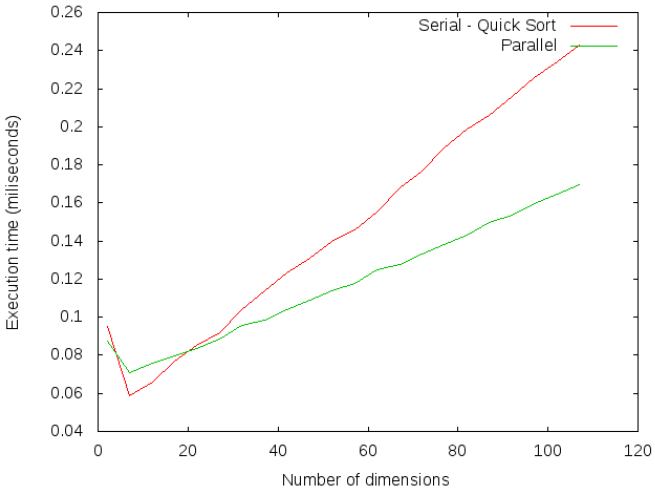

Fig. 5. Comparison of execution time wrt number of dimensions

Its superior floating-point computation capability and low cost appeal to the medium-sized business and individuals. For example, the computing power of a Tesla C1060 card (933 GFLOPS/s) is equivalent to a medium-sized SMP. Problems that used to require a cluster to process can now be solved on a desktop.

References

[1] Kuang, Quansheng, and Lei Zhao. A practical GPU based kNN algorithm. *International symposium on computer science and computational technology (ISCSCT)*. 2009.

[2] Liang, Shenshen, et al. A CUDA-based parallel implementation of K-nearest neighbor algorithm. *Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2009. CyberC'09. International Conference on. IEEE, 2009.