K-Nearest Neighbors (KNN)

Parallel Programming & Architectures

Consideration

- ✓ Your code is automatically graded using a script, and therefore, if your file/folder names are wrong you will receive a grade of **zero**. Please read and follow the instructions carefully. Common mistakes include
  - Different file or folder names
  - Different formatting of input or output
  - Not paying attention to case sensitiveness of C++ and Linux
- ✓ Go to the folder ~/the/project/ in your home directory on the server and put your codes in this directory and remove any compiled binaries and test cases.
- ✓ Make sure your code compiles and runs without any error **on the server**. Your grade will be **zero** if any compile or runtime error occurs on the server. **Any!**
- ✓ The provided test cases, examples and sample codes (if any) are only to better describe the question. They are **not** meant for debugging or grading. It is your responsibility to think of and generate larger and more complex test cases (if necessary) in order to make sure your software works correctly for all possible scenarios.
- ✓ Start early and don't leave everything to the last minute. Software debugging needs focus and normally takes time.
- ✓ Just leave your final programs on the server. **Don't** email anything!
- ✓ Your grade is divided into several parts. In all cases, if you miss **correctness** (i.e. your code doesn't satisfy desired functionality), you miss other parts (e.g. speed, coding style, etc.) too. This rule is applied separately for each section of the project. For example, your code might not be correct for K=100 but still you will get your grade for K=1.
- ✓ Talking to your friends and classmates about this take-home exam and sharing ideas are *OK*. Searching the Internet, books and other sources for any code is also *OK*. However, copying another code is **not OK** and will be automatically detected using a similarity check software. In such a case, grades of the copied parts are **multiplied by -0.5**. Your work must be 100% done only by yourself, and you **should not** share parts of your code with others or use parts of other's codes. Online resources and solutions from previous years are part of the database which is used by the similarity check software.

K-Nearest Neighbors (KNN)

Grading (100):
Report: 15
Correctness: 25
Speed: 60
You will receive the speed grade only if the result is correct.

K-Nearest Neighbors (KNN) is one of the most well-known learning algorithms. Nearest neighbor Search (NNS) is defined as a problem of finding the closest point among references to a query point q. The definition is shown below.

$$nearestneighbor = \underset{x}{argmin}\, d(q, x)$$

Where d is a distance function, and x is a vector in the D-dimensional reference points. The KNN problem is defined similarly. It finds the points p_1, p_2, …, p_k where $d(p_1; q) \leq d(p_2; q) \leq \ldots \leq d(p_k; q)$, and p_1 is the nearest point to q. For more detail, read these articles:

- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

To find the k-nearest neighbors of a query point q, the algorithm can be divided into two major stages. First, the distance between q and each reference vector should be calculated. Second, the distance array should be sorted in ascending order to find the k points which have the least distance from q. However, when k = 1, i.e., for finding the nearest neighbor, there is no need to sort the array. In this case, finding the minimum distance can solve the problem.

In this project, you have 1,000,000 reference vectors which have 128 dimensions. The test query consists of 10,000 vectors with similar dimensions. The distance functions are Manhattan distance (L1-norm), Euclidean distance (L2-norm) and Cosine distance (similarity). They are defined for each $a = (a_0; a_1; ...; a_{n-1})$ and $b = (b_0; b_1; ...; b_{n-1})$ vectors as shown below.

$$Manhattandistance = \sum_{i=0}^{n-1} | a_i - b_i |$$

$$Euclideandistance = \sqrt{\sum_{i=0}^{n-1} (a_i - b_i)^2}$$

$$Cosinedistance = \frac{\sum_{i=0}^{n} a_i b_i}{\sqrt{\sum_{i=0}^{n-1} a_i^2} \times \sqrt{\sum_{i=0}^{n-1} b_i^2}}$$

Implement the above KNN algorithm on GPU. You should calculate the three defined distances for each query point q with all reference vectors. Then, an array of 3k elements should be returned. In this array, the first k elements are related to the Euclidean distance. The elements are the indices of the reference vectors which have the least distance with the query point q in

ascending order. The first element is the index of the vector which is nearest to q. In case of equal distance, less index has priority. The remaining 2k elements refer to the Manhatan and Cosine distances. You should write the output array to a file in your project directory (~/the/project/output.ivecs) for evaluation. You can check the correctness of the Euclidean distance with a ground truth file which is provided.

You will earn the speed's grade in comparison to other students, i.e., the student who achieves the best time will receive a full grade. To improve the speedup you can use the following ideas:

- Exploit the shared memory to reduce the main memory access.
- Try to find the best parameters for the number of blocks and threads. You can use the same idea as the blocked matrix multiplication which you had in previous take-home-exams.
- Use an appropriate sort algorithm such as merge sort, even-odd sort, bitonic sort, or a hybrid approach.
- You can use the k-d tree algorithm. For more information, read k-d tree at Wikipedia.

You are also encouraged to use any ideas which are not mentioned above. However, you are not allowed to use the approximate approaches to find the nearest neighbor. Your task is developing a code to find the exact k-nearest neighbors.

You should report all the ideas which you have utilized to improve your code's speed. Write it in a pdf file named knn_studentID.pdf and put it in the project directory (~/the/project/knn_studentID.pdf). The "studentID" should be replaced by your student ID.

Use the provided gpuerrors.h, gputimer.h and knn.cu files to start your work. You can modify any part in knn.cu. For grading, we will execute the code with N = 10; 000 and K = 1, K = 10 or K = 100. N denotes the number of test vectors in the query, and K is the number of neighbors in the KNN algorithm.

Compile: *nvcc -O2 knn.cu -o knn*

Execute: *./knn N K*