

Quincy Wu

Win EE 590

3/14/17

Machine Learning KNN Algorithm

# Executive Summary

This project is designed to implement the k nearest neighbor algorithm using OpenCL kernel. OpenCL is a low level, cross platform environment to execute program with CPU or GPU. This project focus on the GPU, which is powerful for computing highly parallelable code.

KNN algorithm is one of the most widely used machine learning algorithm. Utilizing the parallel component of the KNN algorithm, this project aimed to speed up the computation time.

# Contents

[Executive Summary 1](#_Toc275344217)

[Contents 2](#_Toc275344218)

[Head 1 3](#_Toc275344219)

[Head 1 3](#_Toc275344220)

# OpenCL

OpenCL is a framework developed for computing in both CPU and GPU. It specifies based on C99 for control the platform between CPU and GPU. It provide a detail insight for program execute in kernels. It allows programmer to control and understand the memory utilization. Unlike CUDA (another GPU computing programming language specify for NVidia GPU), OpenCL is extremely portable. It allows programmer to natively program on large range of device. Although it does not optimize for speed, OpenCL is portable between many devices.

Ever since OpenCL is started in 2009, many mathematical problems were simulated using OpenCL kernel. Heavy computing mathematical model which are easily parallelable, are best for using kernel computing. Image Ray-tracing is a good example for computing using OpenCL kernel.

# KNN Algorithm



K nearest neighbor’s (KNN ) algorithm is one of the simplest and widely used machine learning algorithm. Machine learning focus on training the algorithm to do either classification or regression. In this project, a classification technique is studied. KNN is a type of supervised learning. It takes input of large set of training example of known output to analyses and predict the next output of testing data. Providing the function with increasing amount of data would eventually build a best-fit model for all incoming testing data. KNN algorithm consist of 3 parts, distance matrix calculation, sorting, and majority. For each of the testing data, KNN algorithm calculate the distance between every reference point and testing data to build a distance matrix. Then based on the distance matrix, the k nearest reference data is used to calculate the testing data’s class, based on the majority of class of the k nearest data. KNN algorithm is sensitive to local structure of the reference data to provide the best result.

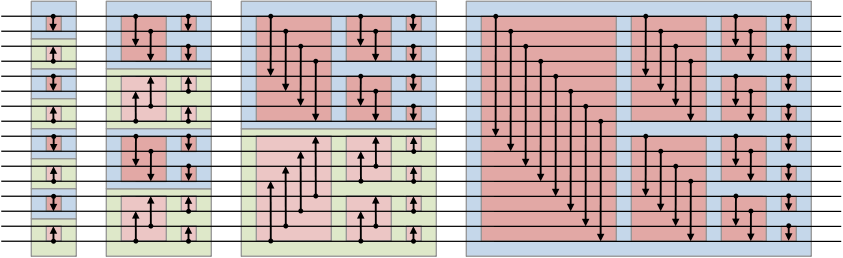
For multiclass KNN algorithm (k greater than 1), KNN algorithm is guaranteed to yield an error rate of no worse than twice the Bayes error rate (minimum error rate of distribution of the data). A special case of the KNN algorithm is 1-nearest neighbor classifier. And in this case the nearest neighbor is used to define the class of testing data.

# Weighting Scheme

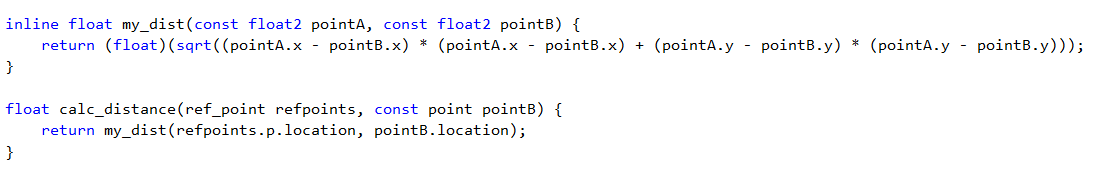
KNN algorithm sensitive to noise. For clustered data and a large number of k (k > 10), a weighting scheme is comely used to reduce the error rate. The weighting scheme generally multiply the k nearest neighbor’s class by 1 over the distance, to increase the importance of closer reference data.

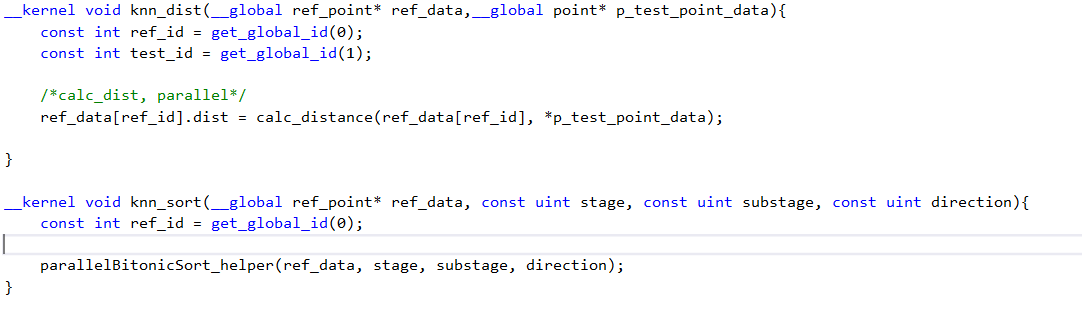
# Sorting (Bitonic)

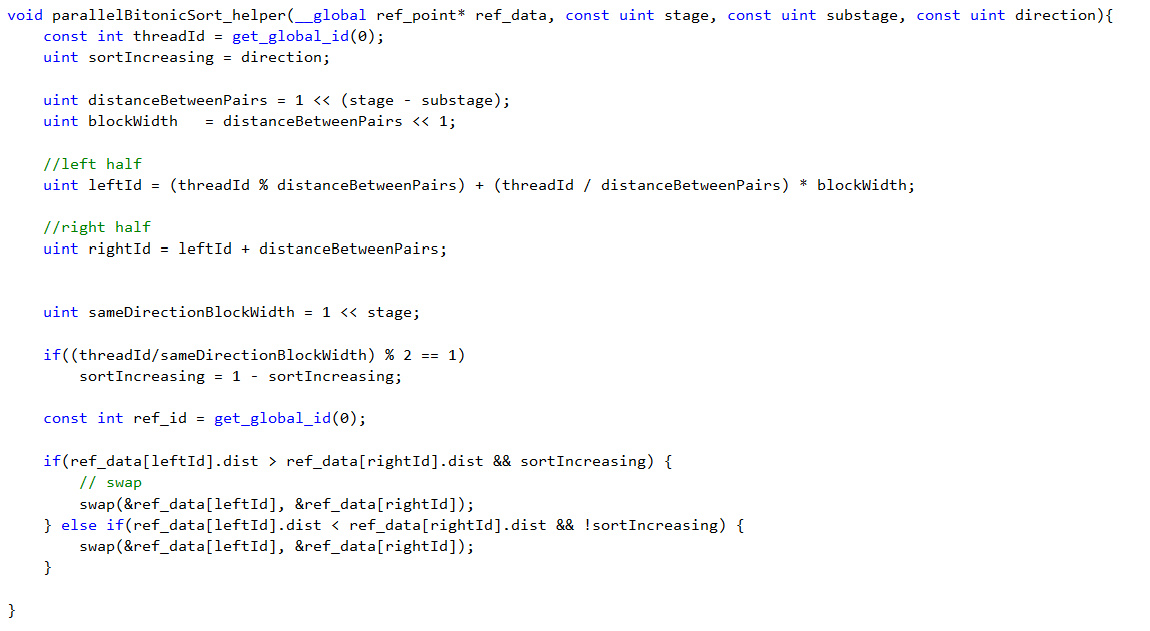
Many sorting algorithm is studied for faster computing speed. This project focus of Bitonic merge sort. Although Bitonic sorting algorithm is a O(n log2 n) and is much slower than radix sort, it is significantly faster than O(n2) bubble sort. “Bitonic sort used by GPUSort does O(n log2 n) work … (compare) to O(n) radix sort.” (Satish) Bitonic sort is designed for parallel computing unlike many other sorting algorithm. It make use of building a sorting network by merging result. An image below provide a good understanding about the connection of Bitonic sort network. Each vertical box’s content can be done in parallels, and its halt and wait for each event in vertical box to finish before moving to the next stage. The Bitonic sort does not really make sense to sort in sequential, so a Batcher’s odd-even merge sort is used in host side for sequential reference.



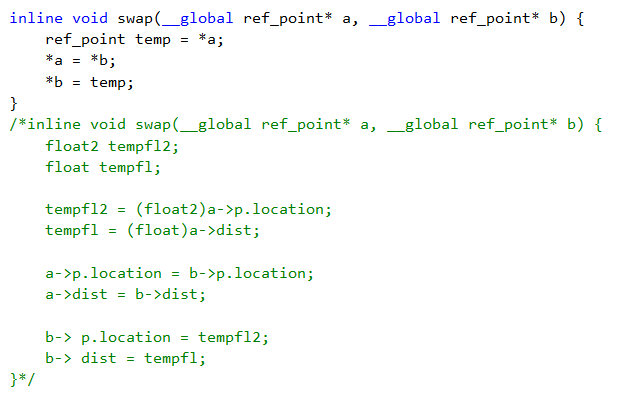
# Sorting (Bitonic)





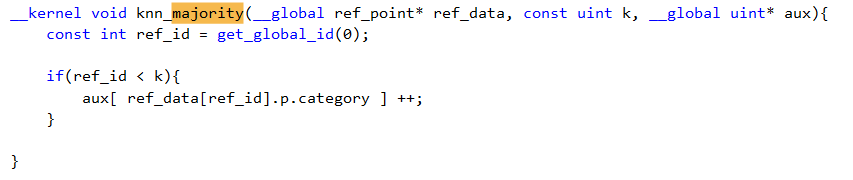


In this project, the calc\_distance and my\_dist is a helper function for knn\_dist kernel. Knn\_dist is the first part of the knn algorithm which calculate the distance matrix. The knn\_dist kernel would calculate the distance between all reference point and one testing point. The main purpose of helper function in here is for code cleanliness. The parallel BitonicSort\_helper function is call by knn\_sort, the second part of KNN algorithm.



# Process & What I learned

In this project, I had a lot of trouble using OpenCL to pass information between host and device and optimize the kernel. I used a struct to represent each point and reference point. Struct provide a clean and simple way for my solution to easily increase number of properties for each data point. Switching from 2d to 3d, or adding 10 more properties for each reference point would be easier to implement using struct. However, visual studio OpenCL does not support struct for kernel side Code Builder. Therefore, I had an attempt to use the Intel OpenCL Code Builder, which support user defined struct. Although it is deprecated, the support for user defined struct is very useful in my case. The most difficult part of this project is to pass piece of memory from host side to kernel side. Although it seems to contain very simple of allocating space, create buffer, and set kernel argument, after receiving the data from host side I had trouble writing global address space variable from kernel back to host side. Variable updated in kernel did not successfully update in host side. Another problem that I had was counting in kernel side. After some research, a global address qualifier is not enough to use as a counter, rather a volatile keyword need to be used. A racing condition were also studied during this project. During the third part of KNN algorithm, the problem I encounter was during the finding of majority category count and update to the host. For example,



If 2 thread received in kernel both tried to update the same location, only one of them would be successfully increase the aux array for the example above. The only solution I found is to move the majority part to host side to compute it sequentially.

Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | Reference size | Test size | k | Category | Local size |
|  | 1024\*1024 | 8 | 10 | 50 | 64 |
| 164.45 ms | 256 | 8 | 10 | 50 | 4 |
| 256.16 ms | 1024 | 8 | 10 | 50 | 4 |
| 484.41 ms | 1024\*16 | 8 | 10 | 50 | 64 |
| 3547.74 ms | 1024\*16 | 64 | 10 | 50 | 64 |
| 15188.04 ms | 1024\*128 | 128 | 10 | 50 | 64 |
| 34223.5 ms | 1024\*512 | 128 | 10 | 50 | 64 |

host

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | Reference size | Test size | k | Category | Local size |
|  | 1024\*1024 | 512 | 10 | 100 | 4 |
| 3.44 ms | 256 | 8 | 10 | 50 | 4 |
| 38.05 | 1024 | 8 | 10 | 50 | 4 |
| 10125.99 ms | 1024\*16 | 8 | 10 | 50 | 64 |
| 574694.8 ms | 1024\*16 | 64 | 10 | 50 | 64 |
| 57482.33 | 1024\*128 | 128 | 10 | 50 | 64 |
| Over 10 mins | 1024\*512 | 128 | 10 | 50 | 64 |

In this project, a lot of effort were made to contain everything in one single kernel. However, the way that I implement the Bitonic kernel, does not allow it to contain everything in kernel. Due to the time limitation of the project, the kernel did not get fully optimize. For the future, optimization such as avoid accessing the global memory every times the kernel is use by utilizing the local memory, using another more efficient sorting algorithm can be done to the kernel to achieve a faster KNN algorithm.

Reference

[Cover TM](https://en.wikipedia.org/wiki/Thomas_M._Cover), [Hart PE](https://en.wikipedia.org/wiki/Peter_E._Hart) (1967). "Nearest neighbor pattern classification". *IEEE Transactions on Information Theory*. **13** (1): 21–27. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1109/TIT.1967.1053964](https://dx.doi.org/10.1109%2FTIT.1967.1053964).