Study of K Nearest Neighbour Applications in Image Processing with Graphics Processing Unit

1Bhavneet Kaur, 2Sonika Jindal

1M.Tech Student, 2Assistant Professor, Department of CSE
SBS State Technical Campus Ferozepur, India
Email: 1sethibhavneet37@gmail.com, 2sonikamanoj@gmail.com

Abstract—The GPU has always been a processor with ample computational resources. It is a powerful graphics engine and a highly parallel programmable processor having better efficiency and high speed that overshadows CPU. It is used in high performance computing system. The implementation of GPU can be done with CUDA C. Due to its highly parallel structure it is used in a number of real time applications like image processing, computational fluid mechanics, medical imaging etc. KNN is the algorithm is the lazy learning method used for finding the points in the data set that are nearest to the query points. In this paper we have shown the various applications where their KNN algorithm is implemented in image processing domain at a high speed as well as giving better performance by using GPU.

Keywords—GPU, CUDA C, KNN.

I. INTRODUCTION

GPU is Graphics Processing Unit used for high performance parallel computing[1]. GPU computing is the use of GPU together with CPU (Central Processing Unit) to accelerate general-purpose scientific and engineering applications. GPU has the following characteristics

- Computational requirements are large. GPUs must deliver an enormous amount of compute performance to satisfy the demand of complex real time applications where many pixels are rendered per second and each pixel requires hundreds of operations.
- Parallelism is substantial. Parallelism adopts graphics pipeline.
- Throughput is more important than latency. GPU implementations of the graphics pipeline prioritize throughput over latency [2].

There are hundreds of processing units present in modern GPUs, that have the tendency to achieve up to 1 TFLOPS[3] for single-precision (SP) arithmetic, and over 80 GFLOPS for double-precision (DP) calculations. Up to 4GB of on board memory and100GB/sec is comprised in the recent high-performance computing (HPC) optimized GPUs [2]. Its parallel architecture and high performance of floating point and memory operations GPU is well suited for many scientific and engineering applications that occupy HPC clusters that leads to their incorporation as HPC accelerators [3][4][5][6]. The GPU, which refers to the commodity off-the-shelf 3D graphics card, is specifically

Figure 1: GPU Architecture
designed to be extremely fast at processing large graphic[4]. Many kinds of computations can be accelerated on GPUs including sparse linear system solvers, physical simulation, linear algebra operations, partial difference equations, fast Fourier transform, level-set computation, computational geometry problems, and also non-traditional graphics, such as volume rendering, ray-tracing, and flow visualization[4][7].

II. ARCHITECTURE OF GPU

The GPU is a processor with ample computational resources. GPU computing is the use of a GPU together with a CPU to accelerate general-purpose scientific and engineering applications [1]. Fig: 1 shows the architecture of GPU where CPU sends tasks and instructions to GPU and all the computations and processing on data is performed by GPU and it further
sends back results to CPU. It can be notified that all the computations are performed by GPU but they are controlled by CPU.

So, in GPU computing:

- GPU code is called DEVICE code.
- CPU code is called HOST code.

GPU has a pipeline structure. The input to the GPU is a list of geometric primitives, typically triangles, in a 3-D world coordinate system. Through many steps, those primitives are shaded and mapped onto the screen, where they are assembled to create a final picture. The specific steps in the canonical pipeline are explained here.

a) Vertex Operations: The input primitives are formed from individual vertices. Each vertex must be transformed into screen space and shaded, typically through computing their interaction with the lights in the scene. Since, typical scenes have tens to hundreds of thousands of vertices, and each vertex can be computed independently, this stage is well suited for parallel hardware.

b) Primitive Assembly: The vertices are assembled into triangles, the fundamental hardware-supported primitive vertices, and each vertex can be computed independently, this stage is well suited for parallel hardware.

c) Rasterization: Rasterization is the process of determining which screen-space pixel locations are covered by each triangle. Each triangle generates a primitive called a Fragment [3] at each screen-space pixel location it covers. Because many triangles may overlap at any pixel location, each pixel's color value may be computed from several fragments [3].

d) Fragment Operations: Using color information from the vertices and possibly fetching additional data from global memory in the form of textures (images that are mapped onto surfaces), each fragment is shaded to determine its final color [1]. Just as in the vertex stage, each fragment can be computed in parallel. This particular stage is typically the most computationally demanding stage in the graphics pipeline.

e) Composition: Fragments are assembled into a final image with one colour per pixel, usually by keeping the closest fragment to the camera for each pixel location.

The programmable units of the GPU follow a single program multiple-data (SPMD) programming model which is well suited to straight-line programs, as many elements can be processed in lockstep running the exact same code [2]. GPU is a type of highly parallel, multi-threaded and multi-core processor. GPGPU which is General Purpose GPU is quite suitable for computing intensive data parallel. CUDA was released in Jun. 2007 by NVIDIA and in Dec 2008, Khronos Group released OpenCL1. AMD launched ATI Stream SDK v2.0 Beta was released in Aug. 2009 that supported X86 processor. Open CL is an open standard used by many processors [8].

III. CUDA

CUDA is Compute Unified Design Architecture. It was introduced by NVIDIA in November 2006. NVIDIA’s CUDA is a general purpose scalable parallel programming model for writing highly parallel applications. It provides several key abstractions – a hierarchy of thread blocks, shared memory, and barrier synchronization. This model has proven quite successful at programming multithreaded many core GPUs and scales transparently to hundreds of cores: scientists throughout industry and academia are already using CUDA to achieve dramatic speedups on production and research codes. The nVIDIA CUDA technology [12] is a fundamentally new computing architecture that enables the GPU to solve complex computational problems.

Compute Unified Device Architecture (CUDA) technology gives computationally intensive applications access to the processing power of nVIDIA graphics processing units (GPUs) through a new programming interface. So, CUDA toolkit provides complete software development for programming CUDA-enabled GPUs [13]. CUDA provides a platform for accessing GPU through standard programming language CUDA C. A number of processor cores are required to execute the compiled CUDA C program. It provides general purpose computing on GPU. CUDA C provides the language integration so that device function calls looks much similar to the host function calls [9]. Following are the restrictions considered in the device pointers applied in CUDA [10]:

- The pointers allocated with cudaMalloc () can be to read or write memory from code that executes on the device.
- The pointers allocated with cudaMalloc () can be passed to functions that execute on the host.
- The pointers allocated with cudaMalloc () cannot be used to read or write memory from code that executes on the host.

CUDA architecture constitutes Grids and Blocks. The grid is composed of multiple threads of the main kernel running parallel and each grid further has a number of blocks that contain the threads and the shared memory [9]. The NVIDIA CUDA [11] is a computing architecture that enables the GPU to solve complex computational problems. Compute Unified Device Architecture (CUDA) technology gives computationally intensive applications access to the processing power of NVIDIA graphics processing units (GPUs) through a new programming interface. Software development can be strongly simplified by using the standard C language.
The CUDA Toolkit is a complete software development solution for programming CUDA-enabled GPUs. This Toolkit includes standard FFT and BLAS libraries, a C-compiler for the NVIDIA GPU and also a runtime driver. CUDA technology is currently supported on the Linux and Microsoft Windows XP operating systems [12]. So, GPU can be accessed with the help of CUDA C language.

**IV. KNEAREST NEIGHBOUR ALGORITHM**

KNN is known as K Nearest Neighbor that is a non-parametric search algorithm applied for classifying the objects based on closest training examples in the feature vector. Here the function is only approximately locally and all computation is deferred until classification. GPU can accelerate the process of the KNN search using NVIDIA CUDA. KNN is also known as “Exhaustive Search” method. This is a problem encountered in many graphics and non-graphics applications [15].

Let \( R = \{ r_1, r_2, \ldots, r_m \} \) be a set of \( m \) reference points in a \( d \) dimensional space, and let \( Q = \{ q_1, q_2, \ldots, q_n \} \) be a set of \( n \) query points in the same space. The \( k \) nearest neighbour search problem involves searching the \( k \) nearest neighbours of each query point \( q_i \) in \( Q \) in the reference set \( R \) given a specific distance. The distance referred may be Euclidean distance, Manhattan distance or Mahalanobis distance [14].

The major steps in the algorithm are as follows [19]:

1. Compute the distance between \( q \) and the \( m \) reference points of \( R \);
2. Sort the \( m \) distances;
3. The \( k \)-nearest neighbours of \( q \) are the \( k \) points of \( R \) corresponding to the \( k \) lowest distances. The output of the algorithm can be the ordered set of these \( k \) distances, the set of the \( k \) neighbours (actually their indices in \( R \)) ordered by the increasing distances, or both.

Fig. 2 shows an example of KNN search where \( K=3 \) in which black dots show set of some reference points and red cross is the query point. Linear scan is the best strategy adopted by KNN [16]. The KNN search is usually slow because it is a heavy process. The computation of the distance between two points requires many basic operations. The resolution of the KNN search grows polynomially with the size of the point sets [15]. The major problem with this algorithm is its huge complexity that is \( O \left( n^m d \right) \) for the \( n \) distances computed and \( O \left( nm \log m \right) \) for the \( n \) sorts performed [14]. So, KNN algorithm is highly parallelizable with GPU where an attempt is made to reduce the computed distances.

KNN uses the Brute Force Search method. It finds its application in many fields like Entropy estimation, content based image retrieval, data mining, clustering and indexing.

**Figure 2: An Example of KNN**

**Applications of KNN in Image processing with GPU**

- Image Searches
- Image Classification
- Spatial Matching
- Image Retrieval

**Figure 3: Applications of KNN with GPU**

image segmentation, statistics, biology etc. In this algorithm, to search for the \( K \) nearest neighbours of a query point \( q \), the distance of the \( K \)-th nearest searching neighbour to \( q \) defines the minimum radius required for retrieving the complete answer set [17].

Fig. 3 show the names of the techniques covered in this work. In this paper we have given a review of different techniques of using KNN with Graphics Processor Units with respect to image classification, image retrieval, spatial matching and Image searching.

A. Image Searching

CUDA C language can be put into use for the implementation of the Brute Force method. The same can also be implemented in MATLAB and KDT-C and it has been proved that the implementation done in CUDA is 120 times faster than BF-MATLAB, 100 times faster than BF-C, and 40 times faster than KDT-C [14]. Due to the huge complexity of the brute force algorithm it needs to be implemented faster so as to reduce the computation time. For this purpose GPU programming is adopted to perform KNN for similarity search. It yields 150 times speed up in comparison to tree based approaches on synthetic data. It also results into 75 times speed up for finding similar patches in images and 70 times faster for texture synthesis [15]. KNN is also used for similarity search in another work where KNN is solved on GPU using efficient metric space techniques. It basically performs search on systems that solve large amount of queries. Here partial ordering of objects is done based on parallel priority queues so as to reduce amount of work. GPU helps achieving considerable speed ups for the corresponding values of \( K \), like 14x4 speed up for \( K=8 \) and 17x speed up for lighter index [16].

There is another work where KNN is used for searching process and due to its high parallelized nature, GPU is used along with it. The parallel implementation is done using the API NVIDIA CUDA and composed of two kernels where first kernel computes the distance between \( n \) query points and \( m \) reference points and stores in matrix of size \( m \times n \). The second kernel sorts

**ISSN (Print): 2319-2526, Volume -3, Issue -3, 2014**
those distances. Sorting is the improved form of insertion sort. It has also used CUBLAS implementation to improve the efficiency. KNN with CUDA gives 64x speed up and with CUBLAS it gives 189x speed up [17].

Spaghettis structure can be used for multi-core and many-core processors based parallel platforms in which the implementations can be evaluated by using two different databases, Spanish dictionary and colour histograms. The load of work can be reduced by arranging the current objects into knn set and keeping their track. While performing all the implementations, the brute force implementation included that the behaviour of the structure in both metric spaces is similar, and the results of speedup came out to be between 1.87 and 3.94 for multi-core implementation, and between 2.08 and 14.04 for the GPU-based platform. To consider the best implementations one can obtain a maximum speed up of 3.17 for multi-core and 9.84 for GPU, both case obtained for the Spanish dictionary, which uses a more expensive computational cost distance function [18].

B. Image Classification

KNN is applied for classification in pattern recognition and machine learning [19].

It can be implemented using a data segmentation method in which there is a train data set containing ‘n’ samples and a test data set containing ‘m’ samples, both are d-dimensional. The training data set is restored in memory as \(n \times d\) matrix and \(m \times n\) matrix for test data set. The result set C, which contains all the distances between each pair of points in A and B, is described in a \(m \times n\) matrix. So the element in data set C which is located in column x and row y, presents the distance between the vector in A whose row number is taken x, and the vector in B whose row number is taken y. For sorting purpose CUDA based Radix sort method is used [19]. On using this method a speed up of 32.61X is obtained as compared with CPU algorithm and 14.98X with ANN-Brute method for a1a dataset and speedup of 34.91X is obtained as compared with CPU algorithm and 15.36X with ANN-Brute method for a2a dataset.

There is another approach called SVM-KNN that is Support Vector Machine-K Nearest Neighbour technique where classification is done based upon visual object recognition including shape and texture. It basically finds the close neighbour to a query sample points and train a local SVM that mainly stores and preserves the distance function on the collection of neighbours. This step supports the major speed up and can also be implemented on GPU to obtain better performance [20].

C. Spatial Matching

The spatial matching can be processed quite loosely or strictly. The spatial consistency is voted by local neighbours [21]. This method is robust to viewpoint changes, but its loose spatial constraints may lead to false matches. But in affine estimation method, the affine transformations are estimated between the query object and candidate images like the LO-RANSAC [22] based spatial matching method can verify spatial configuration strictly.

KNN is used in the ACN-RANSAC method for spatial matching. In this method it performs the matching of the ACN (affine covariant neighbours) corresponding local regions and estimates affine transformation from a single pair of corresponding local regions. It uses the threshold distance that is equal to the \(K_{th}\) nearest distance and also uses Mahalanobis distance covariant with affine transform. The affine covariant local regions have been detected by local region detectors such as the MSER, Harris-Affine and Hessian-Affine detector[24]. The SIFT descriptors [23] have been mapped to the VWs in the codebook which is created by standard k-means or k-means clustering. In addition, a small amount of candidate images with a high probability containing the query object have been obtained based on VWs frequencies [24]. To speed up the spatial matching process, we implement ACN-RANSAC on modern GPU with CUDA efficiently by considering load balancing and optimizing memory accesses [24].

There is a work which used a search area from nearest neighbours (KNN) of each match, and each local region which also matches within this area casts a vote for that match [21]. KNN can also contribute in Spoken Term Detection where it uses query detection framework. The Parallel spoken query detection system can be used that consists of four CUDA kernels. The first two kernels correspond to implementation of a parallel lower-bound estimate, while the last two kernels are used for parallel DTW calculations [25]. The implementation works as follows:

1. Lower bound estimate.
2. Sorting lower bound estimates.
3. KNN search.

Kernal 1 Absolute distance matrix calculation.
Kernal 2 DWT alignment.

After the lower-bound estimates have been computed, each speech segment \(S_j\) is associated with a lower-bound distance estimate \(LB_j\). Since the number of the target speech segments can potentially be very large, a CUDA library called Thrust is used to perform high-performance parallel sorting of these lower-bound estimates [25]. When sorting is complete, the KNN search starts from the speech segment with the smallest lower-bound estimate and calculates the actual DTW distance.

KNN algorithm is also implemented in quantifying shape differences/similarities between pairs of magnetic
resonance (MR) brain images. Here it can be done by performing optimal matching of a large collection of features, using a very fast and hierarchical method from the literature, called spatial pyramid matching (SPM). The similarity between two brain MR images relies on hierarchical feature matching using SPM [26]. The anatomical shape can be obtained from the edges [26]. One can use two basic methods for this implementation:

1. Feature Extraction and Codebook Construction
2. Shape Similarity using Spatial Pyramid Matching.

The work used a GPU implementation of LDDMM and helped to obtain 100x speed up over MATLAB [26].

D. Image Retrieval

KNN can also be used for image retrieval purposes. In content-based image indexing, the process is to perform a similarity measure between images that matches – or at least is close enough to - our perception of their similarity. Then, database images can be simply ranked in increasing order of their similarity to the reference image for a query-by-example task [27].

Kullback-Leibler divergence is a method that is applied in a multi-scale feature space where the feature space consists of inter/intra-scale and inter-channel patches of Laplacian pyramid coefficients for colour images. For the case of colour images, the statistical dependencies are considered amongst the three colour channels; hence patches of coefficients are also inter channel. This approach implies to deal with a high-dimensional statistical description space. To estimate the KL divergence in the k-th nearest neighbour (KNN) framework i.e., adapting to the local sample density and directly from the samples [27]. The parallel implementation of the KNN searching on Graphic Processing Unit (GPU) is done so as to increase the computing time for queries on a large dataset [27]. It is based on brute-force approach and was written in the CUDA. The technique was NVIDIA GeForce 8800 GTX graphic card and allowed to compute a single similarity measure in less than 0.2 seconds on average.

For image retrieval with KNN and GPU affine covariant neighbors (ACN) technique of spatial matching can also be used [28].

There is another approach where the preferences of the customers are retrieved by using a recommendation system which is a software system to predict customers' unknown preferences from known preferences [29]. In a recommendation system, customers' preferences are encoded into vectors, and finding the nearest vectors to each vector is an essential part. This vector-searching part of the problem is called a k-nearest neighbor problem. It can be computed on multiple graphics processor units (GPUs). The algorithm consists of two parts: an N-body problem and a partial sort. For the partial sort, there has been used a novel GPU algorithm which is effective for small k. In the partial sort algorithm, threads are used to access the heap in parallel with a low cost of synchronization. Both of these two parts of the algorithm utilize maximal power of coalesced memory access, so that a full bandwidth is achieved. It is also proved with an experiment that when the size of the problem is large, an implementation of the algorithm on two GPUs runs more than 330 times faster than a single core implementation on a latest CPU [29].

V. CONCLUSION

So, from this paper we have encountered the major use of the GPU that is to obtain higher speed ups and better performance. We have given a review of different methods that can be opted to implement KNN in parallel by using GPU in image processing domain. We have shown that KNN finds its application in the high performance computing where the distance between two points (features) is needed to be found and on basis of which the one with the shortest distance as compared to the threshold distance is chosen as the best match.

ACKNOWLEDGMENT

A special word of appreciation should be given to Mr. Sarabjeet Singh, Assistant Professor, CSE Department, Shaheed Bhagat Singh State Technical Campus, Ferozepur, Punjab, INDIA for his valuable and constructive suggestions during the planning and development of this work. A deep gratitude must also be expressed to Dr Satvir Singh, Associate Professor, Shaheed Bhagat Singh State Technical Campus, Ferozepur, Punjab, INDIA for his patient guidance, enthusiastic encouragement and useful critiques of this task.

REFERENCES


ISSN (Print): 2319-2526, Volume -3, Issue -3, 2014