# PARALLEL kNN ON GPU ARCHITECTURE USING OpenCL

**V.B.Nikam[1], B.B.Meshram[2]**

[1]*Associate Professor, Department of Computer Engineering and Information Technology, Jijabai Technological Institute, Matunga, Mumbai, Maharashtra, India*

[2]*Professor, Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute, Matunga, Mumbai, Maharashtra, India*

## Abstract

*In data mining applications, one of the useful algorithms for classification is the kNN algorithm. The kNN search has a wide usage in many research and industrial domains like 3-dimensional object rendering, content-based image retrieval, statistics, biology (gene classification), etc. In spite of some improvements in the last decades, the computation time required by the kNN search remains the bottleneck for kNN classification, especially in high dimensional spaces. This bottleneck has created the necessity of the parallel kNN on commodity hardware. GPU and OpenCL architecture are the low cost high performance solutions for parallelising the kNN classifier. In regard to this, we have designed, implemented our proposed parallel kNN model to improve upon performance bottleneck issue of kNN algorithm.*

*In this paper, we have proposed parallel kNN algorithm on GPU and OpenCL framework. In our approach, we distributed the distance computations of the data points among all GPU cores. Multiple threads invoked for each GPU core. We have implemented and tested our parallel kNN implementation on UCI datasets. The experimental results show that the speedup of the KNN algorithm is improved over the serial performance.*

***Keywords:*** *kNN, GPU, CPU, Parallel Computing, Data Mining, Clustering Algorithm.*

----------***----------

## 1. INTRODUCTION

In data mining, classification is the problem of identifying set of categories/sub-populations from new observations; based on a training dataset containing observations/instances whose category membership known in advance. The data mining algorithm performs classification is called classifier. Classifier is a supervised learning category. In supervised learning, input set of observations has an effect on output set of observations; the model includes the mediating variables between the input and output variables. kNN uses a continuous data for building classification model. Being the capability of handling large datasets for solving the big problems, kNN has been widely accepted in research and industry applications. Though, kNN has been accepted widely; due to the large datasets, performance became the performance bottleneck for the algorithm. kNN classifier is an instance based learning (IBL) which follows case based reasoning (CBR) approach to solve new problems. This learning approach is a lazy learning. kNN classifier is the most widely used classifier in the data mining domain, and has been accepted for wide range and volume of datasets. A new instance is classified as the most frequent class of its 'K' nearest neighbours.

OpenCL is a generic many core computing programming framework used for parallelizing applications on the GPU's parallel architecture. OpenCL and GPU is most preferred for parallelizing kNN. It is observed that, kNN is embarrassingly parallel in nature and a good candidate for parallelism. In this paper, we have explored the design and experimental results of our parallel kNN on GPU and OpenCL architecture. We have proposed, implemented and tested OpenCL based parallel kNN algorithm on GPU platform. We observed that, our proposed parallel kNN on OpenCL & GPU platform performs well over the sequential performance of kNN. In addition to the performance, the accuracy of our proposed kNN algorithm is achieved to the level of satisfaction.

This paper is organised as below: Section2 discusses the literature survey on kNN classifier, related work in parallel kNN algorithm, OpenCL and GPU technology. Section3 describes our methodology on OpenCL framework for parallel kNN algorithm. Section4 discusses on result analysis. We have observed the results for scaleup on performance. Section5 is the conclusion of our work and the paper.

## 2. LITERATURE SURVEY

The basic process of sequential kNN algorithm is as follows: First, the data pre-processing phase is to initialize the labelled dimensional training dataset as well as the test dataset to be classified. Second, select one test point in the test dataset and calculate the distances between it and each point in the train dataset. The next phase is to sort the results of distances computed, and find out 'k' smallest results. The fourth step is to determine the class label of the test point by the election result of 'k' points. Thus, selecting another point in the test dataset and repeat the process to classify the test dataset[5].

The pseudo-code for the sequential kNN algorithm is as follows,

Algorithm: kNN
Input: *Dataset, D = {(x1,c1), . . . , (xN,cN)},*
*Input query t = (x1, . . . ,xn), k- number of neighbour.*
Output: *class 'c' to be identified for new instance of dataset 't'*

Begin
*For each labelled instance ($x_i$, $c_i$)*
*calculate $d(x_i, x)$*
*Order $d(x_i,x)$ from lowest to highest, (i = 1, . . . ,N)*
*Select 'K' nearest instances to x: $D_x^K$*
*Assign to x the most frequent class in $D_x^K$*
End

If we have 4 test cases and 12 training points, then in traditional algorithm we visit each training point 4 times and will be perform total 48 serial operations. Thus, for 'm' test cases, and 'n' training points there would be O(mn) time complexity. The time complexity is very high if test cases and training points become very large in volume. This has created the need to parallelize operations for kNN algorithm. The compute intensive operational part of kNN algorithm taken up for the parallel and simultaneous operations, can lead to reasonable reduction in compute time. The optimal value for 'k' identified in the range of 3 to 10. In general, large 'k' value is more precise as it reduces the overall noise.

kNN has been used in statistical methodologies and pattern recognition since beginning. 'k' nearest neighbours measured by a distance function. Here we have used Euclidian distance. The Euclidian distance measured between two points, 'a' and 'b', is shown below in equation (1).

$$d(a,b) = \sqrt{\sum_{i=1}^{k}(a_i - b_i)^2} \quad \text{.....(1)}$$

In addition to Euclidian distance, there are Manhattan distance, Minkowski distance are also used for measuring distances. These distance measures are only used for continuous data variables, however, in case of categorical variables the Hamming distance must be used. If there is continuous and categorical variables exist in the dataset, standardization of the numerical variables between 0 and 1 is done to achieve same type data values for the entire dataset. This is also called as normalization of dataset.

## 2.1 OpenCL Framework

Due to tremendous computing power, General Purpose computing on GPU (GPGPU) has been widely adopted by the developers and researchers [7]. Khronos group has proposed OpenCL (Open Computing Language) framework for programming many core architecture [8]. OpenCL is an open industry standard for general-purpose parallel programming. OpenCL device 'kernels' contains computation to be carried out in parallel. In order to achieve parallel computation, many work-items, work groups, etc the components of OpenCL data parallel execution model, as shown in the 0, spawns on multiple data items [9].

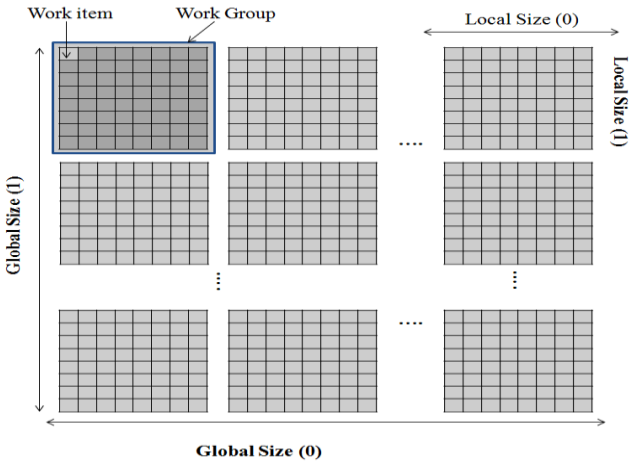



**Fig 1:** Data parallel execution using OpenCL

The data parallelisation task happens through the work items that executes in work-groups as shown in 0 For parallel computing and performance, OpenCL memory hierarchy has four types of device memories viz private, global, constant and local memories. Efficient use of each of them need to be done for high performance achievement [7][8].

## 2.2 Related Work on Parallel kNN

kNN is widely used classification algorithm and very parallel friendly because of number of independent operations. It is required to scan all the objects for any new entry. It is known for lazy learning as all the time dataset needs to be scanned. When training and test dataset size is large, the speed will be quite slow which makes it the best candidate to be processed in parallel. S. Liang , C. Wang, Y. Liu and L. Jian (2009) have focused on parallelizing sorting and distance calculation process due to high compute intensiveness. Distances between the current object and all the data set objects are calculated in parallel. They made two kernels for the same. Sorting Kernel will find the rank of particular object in data set.[1]. Q. Kuang and L. Zhao (2009) proposed parallel scheme for processing testing data set and training data set by dividing computation of result data set. During distance calculation, result data-set is divided into tiles and each tiles of width T is processed by an independent block in parallel. Each block contains TxT thread each of which calculates the single distance[2]. Garcia V. Debreuve E. Nielsen F. and Barlaud M. (2010) suggested CUDA based and CUBLAS based approaches for the parallelism of kNN. The approach except sorting is almost similar to S. Liang et al[1] approach. CUBLAS implementation proved high speed up [3] for performance. G. Nolan proposed improved bubble sort on CUDA for kNN process. they stated iterative version of bitonic sort used for parallelisation[4]. A. Arefin, C. Riveros, R. Berretta and P. Moscato (2011) proposed fast and scalable kNN algorithm on GPU for very large data-sets. In their strategy, they divided reluctant distance matrix into the chunks of size, which can fit into the GPU's onboard memory. This

approach is independent of the size of the distance matrix and overall data size[5]. A. Arefin, et. al. (2012) proposed graph based agglomerative clustering method based on kNN graph and Boruvka's algorithm on GPU[6] .

## 3. PROPOSED METHOLOLOGY

In our proposed parallel kNN algorithm, we have proposed the methodology on OpenCL framework. Our model will performs best on high-end GPU platform, also, the model responds to the multi core commodity architecture. In our parallel kNN model, we compute distance between each training case and all test cases concurrently. Hence, computes all distances in parallel in a step. We extended this concept to calculate distance between all training cases and all testing cases in parallel using many cores. These computations we have taken up on many core GPU platform, and developed kernels in OpenCL to compute the task in parallel.

Finding distance '*d*' for finding nearest neighbour is the crucial and compute intensive task in kNN classifier. We have ported the 'distance' computation on GPU to perform the parallel operation, which has resulted to considerable improvement in kNN performance. The major two kernels work for parallelising the computation on multiple threads on GPU, 1.Distance computation to find the distance between the individual attributes points and 2.Sum up the partial computed distance to find the total sum to get the distance between the points for finding the nearest neighbouring points among the available.

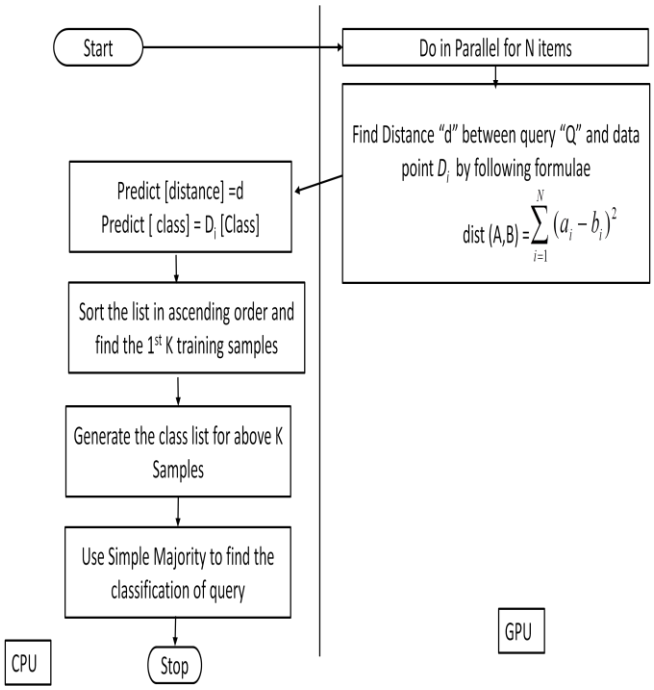



**Fig 2:** Proposed Parallel kNN flowchart

The compute intensive step in kNN is distance computations, which is done on GPU where as others are performed on CPU.

The pseudo code for our parallel kNN algorithm is as follows,

Algorithm: parkNN
Input: Dataset, D = {($a_1$,$c_1$), . . . , ($x_n$,$c_n$)},
Input query, t = ($a_1$, . . . ,$x_n$),
k- number of neighbour.
Output: Class 'c' to be identified for new instance of dataset't'

Begin
On CPU:
1. Load data set to global memory
2. Decide '*k*' for the cluster definition/generation
3. Transfer dataset to GPU memory

On GPU:
4. For all labelled instance ($a_i$, $c_i$), compute $d(a_i, a)$

On CPU:
5. Sort $d(a_i,a)$ from lowest to highest, ( i = 1, . . . ,n)
6. Select '*k*' nearest instances to x: $D_x^k$
7. Assign to x the most frequent class in $D_x^k$

End

Distributing difference computations on many core GPU performs computations faster due to parallelising the difference computations. The following OpenCL kernels demonstrate parallel computation on GPU cores

**Kernel for Parallel difference computation**

```
__kernel void diff(__global const float * query, __global const float *data,
__global const int *nVal, __global float * diff)
{ int id = get_global_id(0);
int indexCent = id % ( nVal[0] * nVal[1] );
int p = id % nVal[0];
int q = (( id / (nVal[0] * nVal[1]) ) % nVal[2] ) * nVal[0];
diff[ id ] = (query[indexCent] - data[ p + q ]) * (query[indexCent] - data[ p + q ]);
}
```

*nVal* contains following parameters,
0 -> #Attributes, 1 -> #Query , 2 -> #Data Points,
# Workers = (#Attributes * #Query * #Data Points)

**Kernel for Parallel Sum Computations**

```
__kernel void Sum(__global const float * data, __global float *sum, __global const int *nVal)
{ int id = get_global_id(0);
int index = id + (( id / nVal[2] ) * nVal[1] );
if( (id % nVal[2]) == (nVal[0] - nVal[2]))
sum[id] = data[index];
else
sum[id] = data[index] + data[(index + nVal[2])];
}
```

*nVal* contains following parameters,
0 -> Data Point Length (DPL), 1 -> DPL / 2, 2 -> Offset , Offset = DPL/2 + DPL%2,
# workers = offset * # Data Points
The parallel computation of the difference computations in two points, A & B, is illustrated in 0, as shown below. The OpenCL kernels written for parallel performance on GPU

device, and are achieving computation methodology as shown in 0.

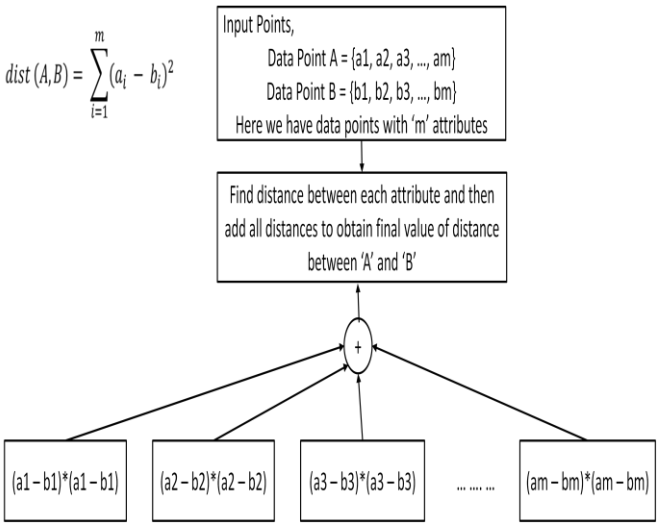



**Fig 3:** Distance calculation between two points in Parallel

Parallel sum kernel as described in kernel 'sum'. Multi-step execution (partial data) for parallel sum calculation generated partial results on each level which further used as inputs to next level of parallel computations.

**Illustrative Example:** We illustrate working of our proposed parallel kNN algorithm with the UCI Balloon dataset, as described in Table1. The attribute values of dataset are transformed into bit-mapped represented datasets. The sample dataset split into training dataset and test dataset with the ratio of 70:30 respectively. For UCI balloon dataset 12 records are taken as training dataset, and 4 instances are taken as test datasets. The training dataset is used to build model, however test dataset used to test the built model for records to classify the query instance. The test data set is shown in Table1, and the training dataset is shown in Table2, as below. The balloon dataset has 4 attributes, 16 records, and is classified in two classes.

**Table: 1** Balloons Dataset. (TEST)

| Sr. No. | Color | Size | Act | Age | Class |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 1 | 1 | 1 |
| 2 | 1 | 2 | 1 | 2 | 2 |
| 3 | 2 | 1 | 1 | 1 | 1 |
| 4 | 2 | 1 | 1 | 2 | 2 |

**Table: 2** Balloons Dataset (TRAINING): Bitmap representation

| Sr. No. | Color | Size | Act | Age | Class |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 2 | 2 |
| 3 | 1 | 1 | 2 | 1 | 2 |
| 4 | 1 | 1 | 2 | 2 | 2 |
| 5 | 1 | 2 | 2 | 1 | 2 |
| 6 | 1 | 2 | 2 | 2 | 2 |
| 7 | 2 | 1 | 2 | 1 | 2 |
| 8 | 2 | 1 | 2 | 2 | 2 |
| 9 | 2 | 2 | 1 | 1 | 1 |
| 10 | 2 | 2 | 1 | 2 | 2 |
| 11 | 2 | 2 | 2 | 1 | 2 |
| 12 | 2 | 2 | 2 | 2 | 2 |

Our proposed parallel kNN algorithm takes into account the same inputs and produces the same output as sequential algorithm, but the kNN operations performed are parallel for performance achievement.

The following steps illustrate the parallel operations of kNN algorithm.

CPU:
1. Load data set to global memory
2. Decide 'k' for the cluster definition/generation
3. Transfer dataset to GPU memory

GPU:
4. For all labelled instance ($a_i$,$c_i$), compute $d(a_i,a)$

CPU:
5. Apply input query on build model for class prediction,
   a. Sort $d(a_i,a)$ from lowest to highest,(i=1,...,n )
   b. Select 'k' nearest instances to x: $D_x^k$
   c. Assign to 'x' the most frequent class in $D_x^k$

The algorithmic steps are illustrated a bit detail as below

**Step1:** Load Data

Load data from CPU memory to GPU global memory. The data is transformed from actual dataset, to bit represented dataset at CPU. The 'bit' represented dataset is then transferred GPU memory.

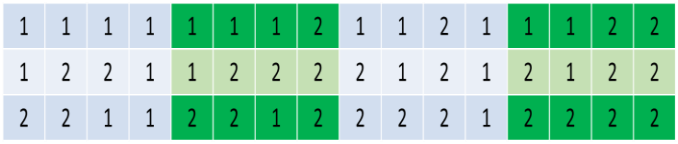
**Step2 :** Define 'k', Let say k=3,

**Step 3:** Transform dataset to GPU memory.

// for distance computations, we used squared Euclidean distance method.

**Step 4:** Make Data vector 'D' and Query vector 'Q'

Vector D (Training Data) contents:




Vector Q (Query Vector) contents:


Computing an Euclidian distance in parallel using openCL kernel __diff( )

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D1-Q1 =1 | D2-Q1 =2 | D3-Q1 =2 | D4-Q1 =3 | D5-Q1 =1 | D6-Q1 =2 | D7-Q1 =3 | D8-Q1 =4 | D9-Q1 =1 | D10-Q1 =2 | D11-Q1 =2 | D12-Q1 =3 |
| D1-Q2 =2 | D2-Q2 =1 | D3-Q2 =3 | D4-Q2 =2 | D5-Q2 =2 | D6-Q2 =1 | D7-Q2 =4 | D8-Q2 =3 | D9-Q2 =2 | D10-Q2 =1 | D11-Q2 =3 | D12-Q2 =2 |
| D1-Q3 =1 | D2-Q3 =2 | D3-Q3 =2 | D4-Q3 =3 | D5-Q3 =3 | D6-Q3 =4 | D7-Q3 =1 | D8-Q3 =2 | D9-Q3 =1 | D10-Q3 =2 | D11-Q3 =2 | D12-Q3 =3 |
| D1-Q4 =2 | D2-Q4 =1 | D3-Q4 =3 | D4-Q4 =2 | D5-Q4 =4 | D6-Q4 =3 | D7-Q4 =2 | D8-Q4 =11 | D9-Q4 =2 | D10-Q4 =1 | D11-Q4 =3 | D12-Q4 =2 |

Here, the parallel model is ready which can be used to predict the input query class. The model is tested in step5. This we have explained in the following four test cases.

**Step 1:** Find Class for each Test case

**Test Case 1**

Distance list is <1, 2, 2, 3, 1, 2, 3, 4, 1, 2, 2, 3>, and their corresponding

Class vector is <1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2 >

Pairing the <Distance vector, Class vector> we get,

<(1,1), (2,2), (2,2), (3,2), (1,2), (2,2), (3,2), (4,2), (1,1),(2,2),(2,2),(3,2) >

After sorting paring vector in ascending order distance wise, we get

<(1,1), (1,2), (1,1), (2,2), (2,2), (2,2), (2,2), (2,2), (3,2), (3,2), (3,2), (4,2)>;

Selecting first 'k' samples we get training samples with corresponding classes as <1, 2, 1>, for k=3.

Thus using majority,we can classify this test case in class 1".

**Test Case 2**

For distance list <2, 1, 3, 2, 2, 1, 4, 3, 2, 1, 3, 2>,

After sorting and selecting first 'k' samples, we get training samples as <2, 5, 10> with corresponding classes as <2,2,2>.

Thus using majority, we can classify this test case in class "2".

**Test Case 3**

Distance list is <1, 2, 2, 3, 3, 4, 1, 2, 1, 2, 2, 3>

After sorting and selecting first k samples we get training samples as <1, 7, 9> with corresponding classes as <1, 2, 1>

Thus using majority we can classify this test case in class"1".

**Test Case 4**

Distance list is <2, 1, 3, 2, 4, 3, 2, 1, 2, 1, 3, 2>

After sorting and selecting first k samples we get training samples as <2, 8, 10> with corresponding classes as <2,2,2>

Thus using majority we can classify this test case in class"2".

We have tested our proposed parallel kNN algorithm on different UCI datasets, as listed in Table-3.

**Table 3:** UCI Classification Dataset

| Sr. No. | Data Set Name | Attributes | Training Records | Testing Records |
|---|---|---|---|---|
| 1. | Balloons | 4 | 12 | 4 |
| 2. | Space Shuttle | 6 | 12 | 4 |
| 3. | Seeds | 7 | 180 | 30 |
| 4. | Met Data | 6 | 432 | 60 |
| 5. | ILPD | 8 | 499 | 80 |
| 6. | CMC Data | 7 | 1400 | 74 |

The experimental results shown that, the model created to predict for target query variable, is working fine and produces almost 85 to 95% accuracy for prediction. As represented in Table4, the accuracy of the prediction depends on the size of the training dataset and the value of the '*k*'. It is also observed that, large the volume of the training dataset, the more mature the model is. Also, large size of '*k*' refines clustering output for the input query, but it does not work always. The best values of the '*k*' for the given dataset has to identify by multiple observation and outputs. We have tested our parallel kNN model on following UCI datasets, categorised for classification only. We have proved our model for performance and classification accuracy, as shown in Table4 and Table5 respectively.

**Table 4:** Parallel kNN Algorithm (Classification Accuracy)

| Data Set Name | 'k' for kNN | Parallel Accuracy |
|---|---|---|
| Balloons | 3 | 100 % |
| Space Shuttle | 5 | 75 % |
| Seeds | 7 | 86 % |
| Met Data | 9 | 95 % |
| ILPD | 7 | 93 % |
| CMC Data | 11 | 96 % |

**Table:5** Parallel kNN Performance on UCI Dataset

| Sr.No | DataSet Name | 'k' Value | Training Records | Testing Rectors | CPU Time | GPU Time |
|---|---|---|---|---|---|---|
| 1 | Balloons | 3 | 12 | 4 | 0.102sec | 9 ms |
| 2 | Space Shuttle | 5 | 12 | 4 | 0.157sec | 13 ms |
| 3 | Seeds | 7 | 180 | 30 | 0.477sec | 21 ms |
| 4 | Met Data | 9 | 432 | 60 | 0.621sec | 56 ms |
| 5 | ILPD | 7 | 499 | 80 | 1.253sec | 76 ms |
| 6 | CMC Data | 11 | 1400 | 74 | 26.72sec | 1384ms |

## 4. CONCLUSION

kNN, the instance-based classifier operate on the premises that classification of unknown instances can be done by relating the unknown to the known using the distance

function. Closer the distance, proximity for the similarity of the class is more. In order to minimize the kNN compute time, we have designed OpenCL based parallel algorithm for many core highly parallel architectures. The results for our OpenCL based parkNN proves that the performance scales up sub-linearly towards the improved performance of parallel kNN. The classification accuracy is not identified with the proportional value of 'k', the better 'k' is to be identified by multiple observations only. However, concerned to performance our parallel kNN, parkNN, we observed satisfactory scale up.

## REFERENCES


[1] Shenshen Liang; Cheng Wang; Ying Liu; Liheng Jian, "CUKNN: A parallel implementation of K-nearest neighbor on CUDA-enabled GPU," *Information, Computing and Telecommunication, 2009. YC-ICT '09. IEEE Youth Conference on* , vol., no., pp.415,418, 20-21 Sept. 2009

[2] Quansheng Kuang, and Lei Zhao, "A Practical GPU based KNN algorithm", Proceedings of the Second Symposium International Computer Science and Computational Technology(ISCSCT '09)

[3] Garcia, V.; Debreuve, E.; Nielsen, F.; Barlaud, M., "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," Image Processing (ICIP), 2010 17th IEEE International Conference on , vol., no., pp.3757,3760, 26-29 Sept. 2010

[4] G. Nolan, Improving the k-Nearest Neighbour Algorithm with CUDA

[5] A. S. Arefin, C. Riveros, R. Berretta, and P. Moscato, "Gpu-fs-knn: A fast and scalable knn computation technique using gpu," Faculty Postgrad Research Poster, The University of Newcastle, Australia, Sept 2011.

[6] Ahmed Shamsul Arefin, Carlos Riveros, Regina Berretta, Pablo Moscato, "kNN-MST-Agglomerative: A Fast and Scalable Graph-based Data Clustering Approach" on GPU The 7th International Conference on Computer Science & Education (ICCSE 2012), July 14-17, 2012. Melbourne, Australia

[7] Franco-Arcega, A.; Suarez-Cansino, J.; Flores-Flores, L.G., "A parallel algorithm to induce decision trees for large datasets," Information, Communication and Automation Technologies (ICAT), 2013 XXIV International Symposium on , pp.1,6, Oct. 30th - Nov. 1st , 2013.

[8] J.Zhu, G.Chen, B. Wu, "GPGPU Memory Estimation and Optimization Targeting OpenCL Architecture", IEEE International Conference on Cluster Computing (CLUSTER), pp. 449-458, Sept. 2012.

[9] Khronos OpenCL working group, "The OpenCL Specification Version2.0", Edited by A. Munshi, Nov. 2013. https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf.


## BIOGRAPHIES

**Valmik B Nikam** is Bachelor of Engineering (Computer Science and Engineering) from Government College of Engineering Aurangabad, Master of Engineering (Computer Engineering) from VJTI, Matunga, Mumbai, Maharashtra state, and pursuing PhD in Computer Department of VJTI. He was faculty at Dr. Babasaheb Ambedkar Technological University, Lonere. He has 12 years of academic experience and 5 years of administrative experience as a Head of Department. He has one year of industry experience. He has attended many short-term training programs and has been invited for expert lectures in the workshops. Presently he is Associate Professor at department of Computer Engineering & Information Technology of VJTI, Matunga, Mumbai. His research interests include Scalability of Data Mining Algorithms, Data Warehousing, Big Data, Parallel Computing, GPU Computing, Cloud Computing. He is member of CSI, ACM, IEEE research organizations, and a life member of ISTE. He has been felicitated with IBM-DRONA award in 2011.

**B.B. Meshram** is a Professor and Head of Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute, Matunga, Mumbai. He is Ph.D. in Computer Engineering. He has been in the academics & research since 20 years. His current research includes database technologies, data mining, securities, forensic analysis, video processing, distributed computing. He has authored over 203 research publications, out of which over 38 publications at National, 91 publications at international conferences, and more than 71 in international journals, also he has filed eight patents. He has given numerous invited talks at various conferences, workshops, training programs and also served as chair/co-chair for many conferences/workshops in the area of computer science and engineering. The industry demanded M.Tech program on Network Infrastructure Management System, and the International conference "Interface" are his brain childs to interface the industry, academia & researchers. Beyond the researcher, he also runs the Jeman Educational Society to uplift the needy and deprived students of the society, as a responsibility towards the society and hence the Nation.