International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

A Predictive Model for Cancer Detection using CUDA K-Means

M. Vishnu Prasath[1], T.Shankar[2], M.Jagan[3], V. Vijayaprasad[4], S.Thanghavel[5], Dr.A.Kunthavai[6]

[1,2,3,4,5]Student, Computer Science and Engineering, Coimbatore Institute of Technology (CIT), Coimbatore, Tamil Nadu, India

[6]Associate Professor, Computer Science and Engineering, Coimbatore Institute of Technology (CIT), Coimbatore, Tamil Nadu, India

---

Abstract - *Cancer is the uncontrolled growth of abnormal cell divisions in human body. Cancer develops when the body has no control over cell division, thus creates enormous extra cells. These extra cells may form a mass of tissue, called a tumor. This abnormal growth of cells can be of two types benign (Non-Cancerous) and malignant (Cancerous). A major challenge in clinical cancer research is the prediction of prognosis at the time of tumor discovery. Accurate prediction of different tumor types can help in providing better treatment and toxicity minimization on the patients. Analyzing the abnormality leads to tremendous amount of data. Mining these data using powerful data analysis tools is important to obtain the biological knowledge needed to classify the cancer types. Analyzing the biological data needs data analytical system to classify the cancer cells without wet lab. Processing of these multidimensional biological data in sequential form will be time consuming and it will reduce the performance of the analytical system. In order to overcome the disadvantages of sequential processing, parallel processing is used. In this project, parallelism is achieved by processing the data in cores of GPU, because GPU contains several hundreds or thousands of cores. Data will be processed in parallel using these cores and hence increases the throughput of the analytical system. In this way, cancer cells can be detected using parallel processing with the use of CUDA (Compute Unified Device Architecture). K-means clustering algorithm is proposed in CUDA to classify the cell mutations into benign and malignant.*

Key Words: Cancer, Benign, Malignant, GPU, Parallel processing, Mutations, CUDA.

1. INTRODUCTION

Data is tremendously growing in all life aspects resulting in mountains of data. Mining these mountains using powerful data analysis tools is important to obtain the contained valuable information needed to present the decision-making solutions. Biological data mining has become an essential part of a new research field called Bioinformatics. It emphasizes on the genomic and proteomic data analysis. Genome is the complete set of genes of an organism. DNA sequences form the foundation of the genetic codes of all living organisms. All DNA sequences includes basic building blocks called nucleotides. A gene usually comprises hundreds of nucleotides arranged in a particular order. One of the most important areas of medical research is the identification of disease-causing genes, which can improve the process of diagnosis and the treatment of diseases. It is known that certain diseases, such as cancer, are reflected in the change of the expression values of certain genes which control how the cell functions. It is characterized by the uncontrolled division of cells to spread, either by direct growth into adjacent tissues through invasion or by implantation into distant sites by metastasis. This growth behavior of cells causes a lump (tumor) to form rogue immune cells that invade and spreads through the blood and lymph systems to other parts of the body. Recently, cancer researchers have attempted to look at cancer prognosis, which is a foreknowledge of an event before its possible occurrence. There are three prognosis foci, which are cancer susceptibility, cancer recurrence and cancer survivability. The spectrum of cancer types exceeds 100 different tumors, named by the location where cancer first developed or by the type of tissue cell in which they start. However, a set of characteristics are shared among almost all malignancies. Cancer classification of different tumor types is of great importance in cancer diagnosis and drug discovery. A major challenge in clinical cancer research is the prediction of prognosis at the time of tumor discovery. Accurate prediction of different tumor types can help in providing better treatment and toxicity minimization on the patients. The recent development of microarray technology has motivated the simultaneous monitoring of genes and cancer classification using gene expression data. Processing of these kinds of biological data in sequential form will be time consuming for large size of multidimensional vectors and it will reduce the capacity of the system. So, it is better to use parallel processing instead of serial processing. It is always a better way to carrying out processing using parallel threads. Parallelism is achieved by processing the data in cores of GPU. Because GPU contains several hundreds or thousands of cores. Data will be processed parallel using those cores and therefore faster throughput can be achieved. In this way, cancer cells can also be detected using parallel processing by CUDA (Compute Unified Device Architecture). To achieve this parallelism, parallel k-means clustering algorithm is proposed in CUDA to classify the human cells into cancer causing cluster and normal cluster.
The list of surveys carried out are,

In the paper [1]," Parallelization of k-means++ using CUDA", k-means++ algorithm is used to find the initial seeds in k-means algorithm. It is an upgraded version of k-means algorithm. K-means++ is then parallelized using CUDA.

In "Accelerating k-means on the graphics processor via CUDA", [2] optimized k-means algorithm is implemented using CUDA. The algorithm is realized in a hybrid manner, parallelizing distance calculations on the GPU while sequentially updating cluster centroids on the CPU based on the results from the GPU calculations.

The paper [3] "Speeding up k-means algorithm by GPUs", also implements k-means algorithm suitable for both low and high dimensional datasets.

In "GPGPU processing in CUDA architecture", [4] makes comparison of CUDA C/C++ with other parallel programming languages like OpenCL and Direct Compute. It also lists out the common myths about CUDA and how the future seems to be promising for CUDA.

The objective of the project is,

To minimize the time taken for serial processing with large data by implementing parallel processing strategy to achieve data parallelization by executing the algorithms in GPUs with the help of CUDA, because GPU processing minimizes the overhead of CPU and it also supports high complex computations.

3. METHODOLOGY

Cancer is a serious and life-threatening disease. Detection of cancer at early stages is a challenging task. A model for detecting the cancer using parallel processing has been proposed. K-means clustering algorithm is proposed in CUDA to classify the human cells into benign and malignant using GPU cores. The modules of the project are listed and the system architecture is shown in Fig -1.

Initially, cancer dataset is collected from COSMIC database. It contains string data. In pre-processing stage, WEKA tool is used to convert string data to nominal data and open cravat tool is used to extract more detailed information from the existing database as shown in Fig -2. Then k-means clustering algorithm is proposed using CUDA to classify the human cells into benign and malignant. Pillar k-means is the optimized k-means algorithm that optimizes initial centroid selection. Attribute selection property is used to classify clusters based on important attributes. Model evaluation includes computing the accuracy before attribute selection and after attribute selection in both serial and parallel processing.

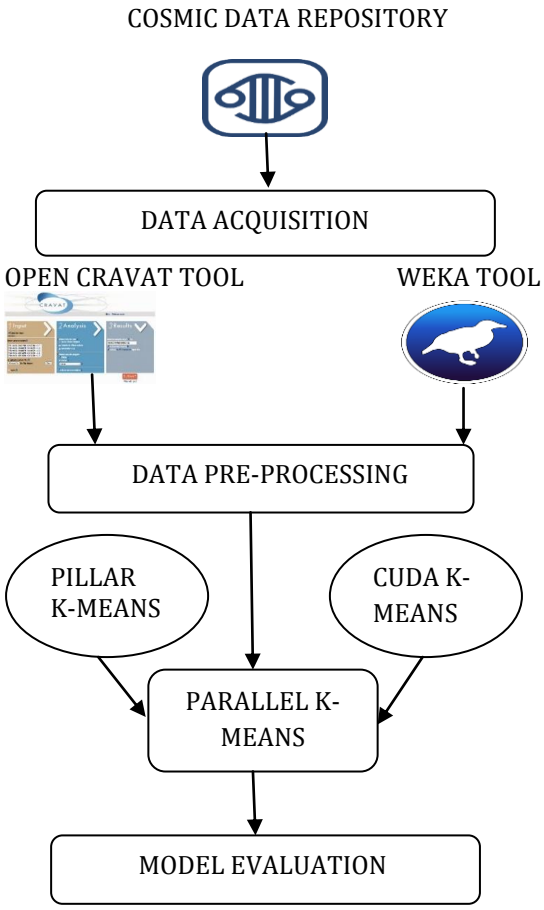

Fig -1: System Architecture

3.1 Data Acquisition and Pre-processing

Cancer dataset is collected from COSMIC (Catalogue of Somatic Mutations) database. It is an online database of somatically acquired mutations found in human cancer. Somatic mutations are those that occur in non-germ line cells that are not inherited by children [5]. In pre-processing stage, missing values and noise values are handled in the dataset. Cancer data consists of protein sequences such as A or C or T or G, position of cell mutation, strand, HUGO symbol etc. Dataset containing attributes such as chromosomes, position, strand, HUGO symbol are available in string format. It is converted to nominal form (binary format) by using WEKA (Waikato Environment for Knowledge Analysis) tool. By using open cravat tool, more detailed information is extracted from existing dataset.

Fig -2: Open Cravat Tool [6]

Fig -3 shows the flowchart of WEKA tool.

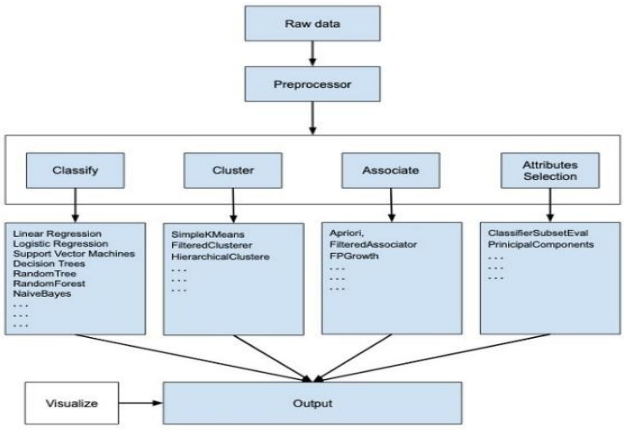

Fig -3: WEKA Flowchart [7]

3.2 Variance Thresholding

Variance thresholding is used for filtering out the required attributes alone from the entire list of attributes available. It is a Feature selector that removes all low-variance features. This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning. Features with a training-set variance lower than this threshold will be removed [8]. Here, the threshold value is set as 0.5.

Hence all the attributes having a variance value less than 0.5 will be eliminated and the remaining attributes alone are returned. It is implemented in python. The Variance thresholding program takes the initial dataset as input and returns a dataset with the attributes having greater variance value alone. Here in initial dataset there are 98 columns and after the variance thresholding, the new dataset has 28 columns.

When both the datasets are executed and the results are compared, both yields the same accuracy. But the time taken is less in the later dataset than the former.

3.3 Parallel k-means for cancer detection

Parallel processing is a mode of operation in which a process is split into parts, which are executed simultaneously on different processors attached to the same computer. It is done by using "Divide and conquer" principle. Process is split into threads and it will be executing in cores of GPU (Graphics Processing Unit) through CUDA (Compute Unified Device Architecture).

3.3.1 K-means clustering

K-means is a clustering algorithm used to classify cancer cells into driver and passenger cells. K-means is parallelized by using Compute Unified Device Architecture (CUDA). It is one of the most popular "clustering" algorithms. It is an unsupervised learning algorithm. K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition *n* observations into K clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. K-Means minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. Better Euclidean solutions can be found using k-medians and k-medoids. The problem is computationally difficult NP-hard; however, efficient heuristic algorithms converge quickly to a local optimum [9]. In order to prevent the algorithm from reaching local optimum, Pillar K-means algorithm is used for initial seed selection.

3.3.2 Pillar k-means

The advanced development of electronic data has brought two main impacts in data clustering algorithm, including how to store big data and how to process this type of data [10]. Pillar K-means cluster is one of clustering algorithm developed from K-means algorithm which is more effective than another similar algorithm. It is used for initial seed selection. It selects optimized initial centroid. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modelling. They both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes. The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by *k*-means classifies new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm [11]. The formula for calculating Euclidean distance d(x,y) is mentioned below:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Where d(x,y) is the euclidean distance between x and y rows.

3.3.3 Steps to calculate k-means clustering

Let X = {x_1,x_2,x_3,……..,x_n} be the set of data points and V = {v_1,v_2,……..,v_c} be the set of centers.

- **Randomly select *'c'* cluster centres**

As a starting point, model should decide how many clusters it should make. First the model picks up K, (let K = 3) data points from the dataset. These data points are called cluster centroids. Now there are different ways you to initialize the centroids, you can either choose them at random — or sort the dataset, split it into K portions and pick one data point from each portion as a centroid.

- **Calculate the distance between each data point and cluster centres**

The model performs calculations on it's own and assigns a cluster to each data point. The model would calculate the distance between the data point.

- **Assign the data point to the cluster centre whose distance from the cluster centre is minimum of all the cluster centres**

The data point will be assigned to the cluster with the nearest centroid.

- **Recalculate the new cluster centre using**

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

Where, 'c_i' represents the number of data points in i^{th} cluster.

'x_i' represents the data points in i^{th} cluster.

'v_i' is the new centroid data point.

- Recalculate the distance between each data point and new obtained cluster centres

- If no data point was reassigned then stop, otherwise repeat the process [12].

3.3.4 CUDA

NVIDIA provides a programming interface known as CUDA (Compute Unified Device Architecture) which allows direct programming of the NVIDIA hardware. Using NVIDIA devices to execute massively parallel algorithms will yield a many times speedup over sequential implementations on conventional CPUs [13]. It is a parallel computing platform and application programming interface (API) model. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units) [14].

In contrast to CPU computer, one GPU contains several hundreds or even thousands of cores as shown in figure 4. It uses high-bandwidth bus (~200Gb/s) connecting the memory on chip to the computing cores and is optimized for parallel calculations, particularly for single instruction multiple data (SIMD) operations. The Fig-4 shows the comparison between CPU cores and GPU cores.

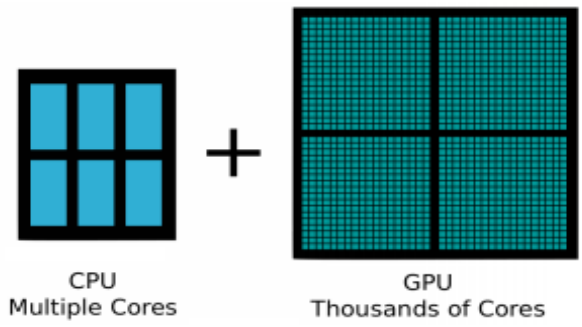

Fig -4: CPU cores VS GPU cores [13][14]

The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming.

The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives such as OpenACC, and extensions to industry-standard programming languages including C, C++ and Fortran. C/C++ programmers can use 'CUDA C/C++'. Fortran programmers can use 'CUDA Fortran' [14]. The processing flow of CUDA is shown in Fig -5.

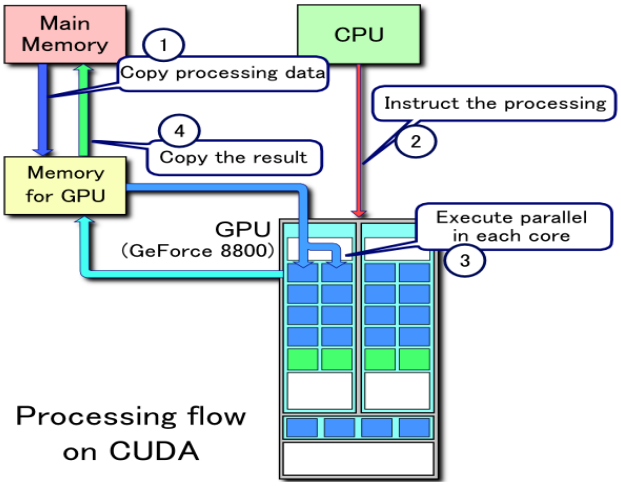

Fig -5: Processing flow of CUDA

Process Flow includes,

- Copy data from main memory to GPU memory.
- CPU initiates the GPU compute kernels.
- GPU's CUDA cores execute the kernel in parallel.
- Copy the resulting data from GPU memory to main memory [14].

In the CUDA processing paradigm (as well as other paradigms similar to stream processing) there is a notion of a 'kernel'. A kernel is essentially a mini-program or subroutine. Kernels are the parallel programs to be run on the device (the NVIDIA graphics card inside the host system). A number of primitive 'threads' will simultaneously execute a kernel program. Batches of these primitive threads are organized into 'thread blocks'. A thread block contains a specific number of primitive threads, chosen based on the amount of available shared memory, as well as the memory access latency hiding characteristics desired. The number of threads in a thread block is also limited by the architecture to a total of 512 threads per block. Each thread within a thread block can communicate efficiently using the shared memory scoped to each thread block. Using this shared memory, all threads can also sync within a thread block. Every thread within a thread block has its own thread ID. Thread blocks are conceptually organized into 1D, 2D or 3D arrays of threads for convenience.

As shown in Fig -6, a 'grid' is a collection of thread blocks of the same thread dimensionality which all execute the same kernel. Grids are useful for computing a large number of threads in parallel since thread blocks are physically limited to only 512 threads per block. However, thread blocks within a grid may not communicate via shared memory, and consequently may not synchronize with one another.

- Threads are grouped into blocks.
- Blocks are grouped into a grid.
- A kernel is executed as a grid of blocks of threads.

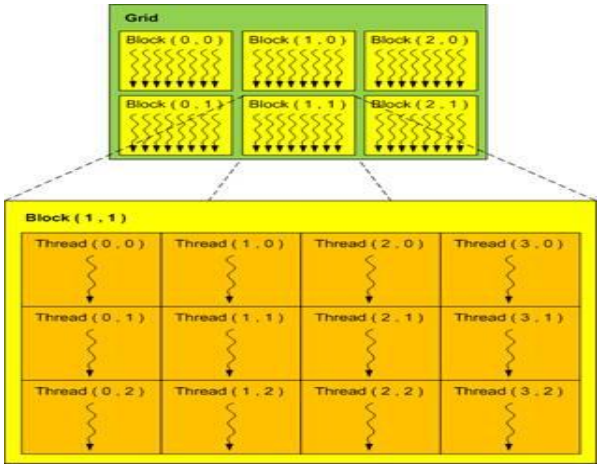

Fig -6: Thread Hierarchy

The above diagram demonstrates the thread hierarchy described. Here, a given kernel contains a 3x2 grid of thread blocks. Each thread block is a 4x3 block of threads, yielding a total of 72 threads executing said kernel [15].

The interface between the host system and the CUDA device(s) is the cuda_value function. It is responsible for allocating memory on the device for the weights, the data, and intermediate results (i.e., the probabilities and the gradient). Host memory is much larger than the device memory, so we must partition the training data into smaller slices (several columns of **X**), transferring each to the device and calculating their contributions to the objective function value and gradient one slice at a time. All four steps (energy through gradient calculation) must be completed on the device for the given slice of instances before the next slice is processed.

The value and gradient are sums over the *N* instances, this is a straightforward process of adding results to accumulator variables. The partial sum for the objective function value is calculated on the device and then accumulated on the host. The partial sums for the gradient matrix are accumulated on the CUDA device and copied back to the host when all the data has been processed. This is because the gradient is calculated using the CUBLAS library, which makes it easy to accumulate matrix products in parallel [16].

4. PSEUDO CODE

Parallel K-Means (P,K)

Input: *a dataset of points P = {P1,…,Pn},a number of Clusters k*
Output: *centers {c1,…ck} Implicitly dividing P into k clusters*
Choose k intial centers C = {c1,…,ck} using pillar K-means
Pillar k-means:
First seed
do
for *i=1,……,N calculate the sum of all data points*
Find the Grand mean (m) of P using the sum
for *i=1,……,N calculate the distance (di) between m and i*
Select the point (c1) corresponding to max (di) as
First seed
*//**end do***
Second seed
do
for *i=1,……,N Calculate the distance(d1i)between c1 and i*
Select the point (c2) corresponding to max (d1i) as
Second seed*.*
*//**end do***
While *convergence has not been met*
do *-> assignment step:*
for *i=1,…..,N* ***do***
find closest ck belongs to C to instance Pi
assign instance Pi to set ck
update step:
for *i=1,….,k* ***do***
set Ci to be the center of mean of all points in Ci
*//**end***

5. EXPERIMENTAL RESULTS

The detection of cancer cell has been achieved through parallel processing with the help of CUDA (Compute Unified Device Architecture) using k-means clustering algorithm. Implementation is done with Windows operating system 32 or 64 bit processor, Visual Studio 2017, NVIDIA CUDA platform, C programming language.

The mutation data for the cancer detection is obtained from the cosmic database. The obtained data is then pre-processed using WEKA and Open CRAVAT tool. The required attributes for the classification are selected using Variance thresholding. The pre-processed data is then classified using CUDA K-means Clustering. The cluster obtained is compared with the attributes of the previously obtained cancer dataset and cluster is classified as benign (cancerous) or malignant (non-cancerous).

The raw data and the pre-processed data are shown in Fig-7 and Fig-8.

Chrom	Position	Strand	Ref. base(	Alt. base(s	HUGO syn	Sequence	Protein se
chr8	57228816	-	C	T	SDR16C5	MS	L31F
chr10	50534350	+	G	A	C10orf71	MS	V1254I
chr12	9317862	-	C	A	PZP	MS	S787Y
chr17	38955885	-	G	T	KRT28	MS	K87N
chr6	1.1E+08	-	G	C	MICAL1	MS	D461H
chr6	38830152	+	G	T	DNAH8	MS	M2076I
chr7	72957910	-	C	T	BCL7B	MS	S78L
chr8	1.39E+08	-	G	A	FAM135B	MS	D787N
chr8	25364913	+	C	T	CDCA2	MS	P911S
chr19	58489941	-	A	G	ZNF606	MS	T703A
chr11	58920660	+	G	A	FAM111A	MS	E507K
chr12	56113352	+	G	C	BLOC1S1	MS	E141Q
chr5	70858273	+	G	C	BDP1	MS	E2557Q
chr8	36663872	+	C	T	KCNU1	MS	S185F
chr15	64204356	-	T	A	DAPK2	MS	V300E
chr19	38379723	-	C	A	WDR87	MS	Q1491K
chr10	16979594	-	C	T	CUBN	MS	P1975S
chr1	51754571	-	C	G	TTC39A	MS	A521G
chr11	28135038	+	C	G	METTL15	MS	Q53E

Fig -7: Dataset before pre-processing

Chrom	Position	Strand	Ref. base(	Alt. base(s	Sequence
8	57228816	0	2	4	1
10	50534350	1	3	1	1
12	9317862	0	2	1	1
17	38955885	0	3	4	1
6	1.1E+08	0	3	2	1
6	38830152	1	3	4	1
7	72957910	0	2	4	1
8	1.39E+08	0	3	1	1
8	25364913	1	2	4	1
19	58489941	0	1	3	1
11	58920660	1	3	1	1
12	56113352	1	3	2	1
5	70858273	1	3	2	1
8	36663872	1	2	4	1
15	64204356	0	4	1	1
19	38379723	0	2	1	1
10	16979594	0	2	4	1
1	51754571	0	2	3	1
11	28135038	1	2	3	1

Fig -8: Dataset after pre-processing

Resultant clusters obtained using parallel K-Means before and after feature selection are shown in Fig -9 and Fig -10.

Fig -9: Parallel k-means before feature selection

Fig -10: Parallel k-means after feature selection

Thus, the time taken for running parallel k-means clustering algorithm before featured selection is greater than running k-means clustering algorithm after featured selection.

Resultant clusters obtained using serial k-Means before and after feature selection are shown in Fig -11 and Fig-12.

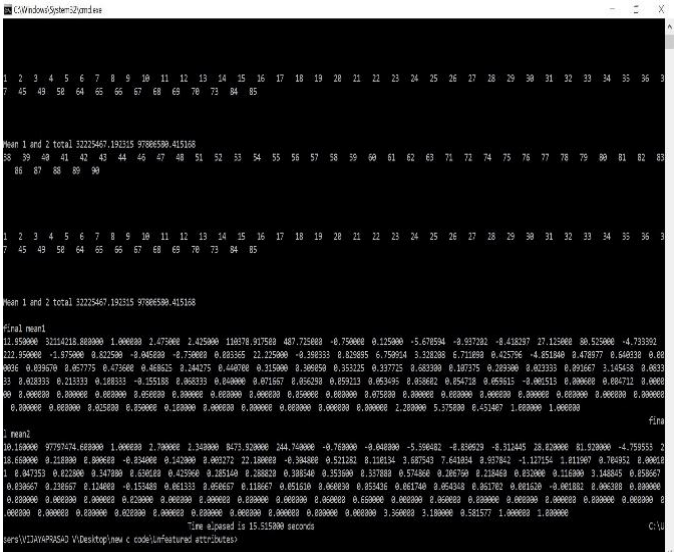

Fig -11: K-means before feature selection

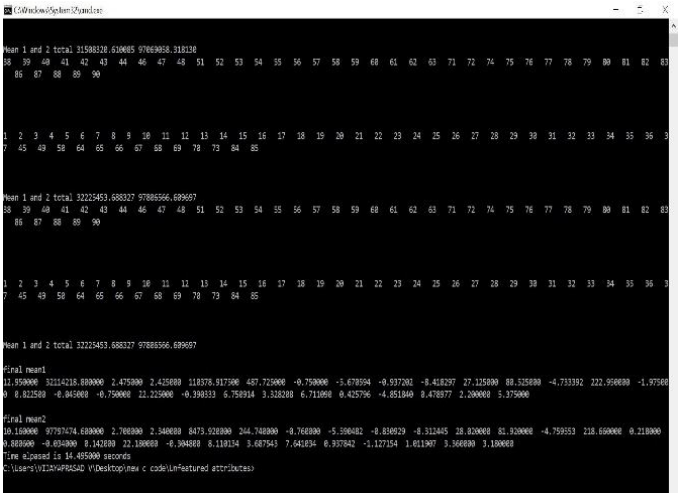

Fig -12: K-means after feature selection

Thus, the time taken for running k-means clustering algorithm before featured selection is greater than running k-means clustering algorithm after featured selection.

Hence, in both CUDA and C, feature selection helps to minimize the runtime of the algorithm.

The time comparison between Serial Processing VS Parallel Processing is shown Fig -13

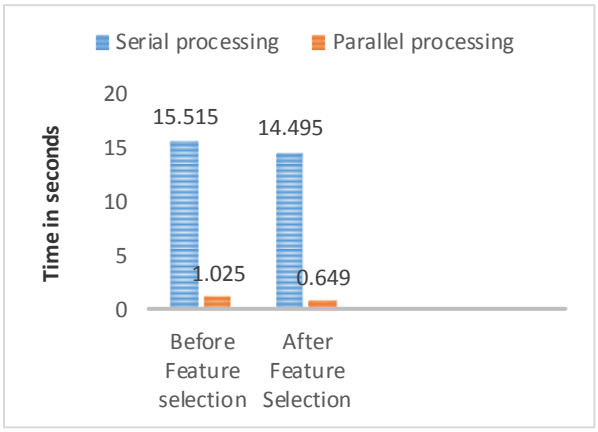

Fig -13: Serial processing VS parallel processing

From the above graph it is incurred that the time consumed by CUDA K-means before feature selection takes 6.6% of time taken by serial implementation and after feature selection 4.5% of time when compared to that of serial processing.

6. CONCLUSIONS AND FUTURE WORK

In this paper, it is shown how to parallelize an algorithm (K-means algorithm) used by many applications on a highly parallel graphics processing architecture. Our results suggest that computing an algorithm in parallel environment (CUDA) takes lesser time than other strategies like serial processing and it also minimizes the overhead of CPUs.

Detecting the cancer at early stage is crucial and essential task now-a-days for medical diagnosis. Data mining techniques provides facility to design and develop a predictive model for identifying cancer cells. This paper shows the implementation of these techniques in parallel programming approach that helps in improving the efficiency of the predictive model.

Future work includes classifying the driver mutations as curable and non-curable.

REFERENCES

[1] Maliheh Heydarpour Shahrezaei, Reza Tavoli, (2019), "Parallelization of Kmeans++ using CUDA", arXiv:1908.02136v1[cs.DC], (https://arxiv.org/pdf/1908.02136)

[2] Mario Zechner, Michael Granitzer, (2009), "Accelerating k-means on the graphics processor via CUDA", In proceedings of IEEE (Vol.1, pp. 7-15) (https://www.computer.org/csdl/proceedings-article/intensive/2009/3585a007/12OmNwlqhS9)

[3] You Li, Kaiyong Zhao, Xiaowen Chu, Jiming Liu, (2013), "Speeding up k-means algorithm by GPUs.",(Vol.79,issue.2,pp.216-229) (https://www.sciencedirect.com/science/article/pii/S0022000012000992)

[4] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, Amit Bawaskar, (2012), "GPGPU Processing in CUDA architecture", (ACIJ,Vol.3,No.1) (http://www.airccse.org/journal/acij/papers/0112acij09)

[5] Wikipedia for COSMIC database information (https://en.wikipedia.org/wiki/COSMIC_cancer_database)

[6] Biostars for open cravat tool informations (https://www.biostars.org/p/354542/)

[7] Tutorialspoint for WEKA tool informations (https://www.tutorialspoint.com/weka/weka_quick_guide)

[8] Scikit-learn for variance threshold informations (https://www.tutorialspoint.com/weka/weka_quick_guide)

[9] Wikipedia for k-means clustering informations (https://en.wikipedia.org/wiki/K-means_clustering)

[10] A.L.Ramdani and H.B.Firmansyah "Pillar K-Means Clustering Algorithm Using MapReduce Framework"., IO012031P Conf .Series: Earth and Environmental Science 258 (2019), doi:10.1088/1755-1315/258/1/012031

[11] Medium for details about Pillar k-means algorithm (https://medium.com/@sourodipkundu8/kmeans-for-beginner-4af96166379e)

[12] Towards data science for details about k-means clustering (https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a)

[13] Game scanning for details about CUDA cores (https://www.gamingscan.com/what-are-nvidia-cuda-cores/)

[14] Wikipedia for CUDA processing flow information (https://en.wikipedia.org/wiki/CUDA)

[15] CUDA.ce.rit.edu for overall CUDA information about thread hierarchy (http://cuda.ce.rit.edu/cuda_overview/cuda_overview)

[16] Science direct for CUDA value information (http://cuda.ce.rit.edu/cuda_overview/cuda_overview)

BIOGRAPHIES

M. Vishnu Prasath[1], pursuing final year Bachelor's degree (B.E) in Computer Science and Engineering, Coimbatore Institute of Technology, Coimbatore, Tamil Nadu

T.Shankar[2], pursuing final year Bachelor's degree in Computer Science and Engineering, Coimbatore Institute of Technology, Coimbatore, Tamil Nadu.

M.Jagan[3], pursuing final year Bachelor's degree (B.E) in Computer Science and Engineering ,Coimbatore Institute of Technology, Coimbatore, Tamil Nadu.

V.Vijayaprasad[4], pursuing final year Bachelor's degree (B.E) in Computer Science and Engineering, Coimbatore Institute of Technology, Coimbatore, Tamil Nadu.

S.Thanghavel[5], pursuing final year Bachelor's degree (B.E) in Computer Science and Engineering, Coimbatore Institute of Technology, Coimbatore, Tamil Nadu.

Dr.A.Kunthavai[6] received BE degree in Computer Science and Engineering from Madurai Kamarajar University, MS degree (by research) in 2008 and PhD degree in 2014 from Anna University, India. Currently, she is working as an Associate Professor in computer science and engineering at Coimbatore Institute of Technology, India. She is a life member of ISTE. Her research interest includes bioinformatics, medical imaging, distributed system and data mining.