2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI) | 978-1-7281-9675-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDABI51230.2020.9325700

Applying Parallel Processing to Improve the Computation Speed of K-Nearest Neighbor Algorithm

Maha A. Alanezi
Big Data Science & Analytics Master Program
Collage of Science
University of Bahrain
Sakhir, Kingdom of Bahrain
mahaalanezi218@gmail.com

Abdulla AlQaddoumi
Department of Information System
Collage of Information Technology
University of Bahrain
Sakhir, Kingdom of Bahrain
aqaddumi@uob.edu.bh

***Abstract*— K-Nearest Neighbor (KNN) is a widely used algorithm to gain an accurate and efficient classification. One of the drawbacks of the algorithm is the time required to calculate the distance for each point. In this paper, the aim is to speed up the KNN algorithm with the implementation of multi threads and multiprocessors to reduce the time to execute the algorithm. There are two medical datasets used to apply the KNN algorithm for the comparison of the sequential and parallel performance. The datasets utilized are Heart Test and Breast Cancer Wisconsin (Diagnostic) Data Sets. The multiprocessing is used to parallelize the computation of the distance of KNN. The parallel KNN outperforms the sequential version with the speedup increase, leading to reduction in time for both the processes and threads.**

Keywords*— K-Nearest Neighbor, Improve Performance, parallel K-Nearest Neighbor.

I. INTRODUCTION

The 21st century is the era with a massive amount of data, and it continues to expand rapidly. The ordinary performance of the computers cannot handle analyzing this massive amount of data and higher intelligence is required to cope with this current growth of the Big Data. Considering the high-performance computing (HPC) while programming and creating software is a must, because it can increase the computation speed and give accurate results with less cost. High performance computing have several types such as Computer clusters, Grid Computing, Cloud computing, Graphical Processing Units (GPU), MIC, and FPGA [1]. The distributed memory programming with multi processors is one of the methods that achieves high performance on GPU level. One way of using the high performance computing to manage the Big Data is to parallelize machine learning algorithms.

One of the most used machine learning algorithms is the K-Nearest Neighbors (KNN) due to the simplicity and efficiency of the classification [2]. KNN is a method that uses the closest point in the training set for classifying the object [2]. The algorithm is considered a simple algorithm, handles noisy dataset and has effective results [3]. However, the algorithm's disadvantage is the difficulty in calculation and the cost of the implementation of the method is high [3]. The algorithm is constructed in parallel to reduce the complexity of the computation and time required to perform the distance calculation for each point in the dataset. Therefore, the aim of this paper is to speed up the KNN algorithm with the implantation of multi threads and multiprocessors to reduce the time to execute the algorithm.

The paper is organized as follows: related work is presented in section 2. The methodology is described in section 3. In section 4, experiments conducted on several datasets are presented. The conclusion is presented in section 5.

II. RELATED WORK

A study by Zhang, Li, and Jestes in 2012 executes the parallel KNN algorithm based on MapReduce with heterogeneous cluster by applying the block nested loop methodology for KNN-joins. The data is divided into equal-sized blocks and then separated to buckets in the Map phase. After that, in the reduce phase, the block nested loop KNN join is performed by reducer for each bucket to be saved as DFS files. The result showed that the partitions and the running time was balanced, and the speedup increased as the cluster gets larger. However, the study results that the communication using H-BRJ is double the H-zkNNJ and as the dimension increase, there is a reduction in recall and precision to reach more than average [4].

Another study by Sismanis, Pitsianis, and Sun in 2012 applying the K-Nearest Neighbors (KNN) for higher dimension with multi core processors in the graphics processing unit (GPU). The parallelization of KNN algorithm in the sorting by truncated bitonic sort (TBiS). The study found that the performance using GPU with TBiS outperforms the sort and select methods [5].

A research by Rajani, McArdle, and Inderjit in 2015 using Tree-Based Data Structures in OpenMP and the Galois framework to execute the parallel k nearest neighbor. There are four categories of threading used in OpenMP to implement KNN: with 1, 4, 8, and 16 threads and Amdahl's law was used to measure the speedup. The research found that the ball trees performs better than k-d trees in higher dimension using MNIST datasets and it is efficient comparing it to Scikit-learn in python. However, as the dimensions decrease the time is faster with a linear speedup for the datasets [6].

Moreover, Pu, Peng, Huang, and Chen in 2015 implemented the KNN algorithm on FPGA and GPU using OpenCL. The study executed the bubble sort after the data is transferred from CPU to FPGA to perform distance calculation and distance ranking in parallel. First, the distance calculation in parallel is implemented to find the distance between each object, then sorting is performed to find the K smallest distance. The result showed that for the computation speed, the GPU performs better. However, if the average is used, Joule the FPGA outperformed the GPU implementation.

978-1-7281-9675-6/20/$31.00 ©2020 IEEE
Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.

Also, compared with traditional GPU, the performance was superior [7].

A study conducted in 2016 with distributed architectures to parallelize K-Nearest Neighbor algorithm. First, the points are split up to equal number in each node to two subsets with every node, including the points that belong to the subset through communication. Then, parallelizing data for split and rearrange the points in a shared memory environment to reach an adequate branch. After that, thread parallelism is used where each thread constructs the K-tree with independent points. Three datasets related to the field of science were used to test the parallel KNN. The first dataset is related to astrophysics, the second dataset is related to plasma physics, and the third dataset is related to particle physics. The study found that, in 48 seconds, the KD-trees is constructed using 50,000 cores and the calculation of KNN is done in 12 seconds. Also, the algorithm outperformed in comparison with the other implementations in Big Data analytics problems and for the shared [8].

A study in 2019 applied parallel KNN algorithm in matching for 3D reconstruction. The research used Nvidia CUDA SDK on GPU device for the higher resolution images. The pipeline is used for real-time feature tracking and the distance computing result is saved in shared memory to save access time. The result shows that it is 10 time faster using parallel KNN then sequential KNN with better efficiency and effectiveness [9].

A research in 2019 used parallel KNN classifer with two methods the data parallism and task parallism. The method of parallism are used in classifing image and model training. Massage passing interface is used in C++ environment. The result shows that data parallelism is more effective than task parallelism due to parallel overhead [10].

Another research in 2020 used cloud services to run KNN in parallel for the purpose of classifying encrypted data. The data parallelism is applied to parallel KNN algorithm. The result shows that the time taken in sequential KNN is 12.02 to 55.5 minutes, but the same data is used in parallel KNN with time recorded is 4.16 minutes [11].

III. PROPOSED ALGORITHM AND METHODOLOGY

These are many reasons to parallelize the KNN algorithm like the complexity in the computation, and the high cost of execution. In order to evaluate the performance of the parallel KNN algorithm, two datasets were used. The first dataset, Heart Test Dataset, was used from Kaggle Machine learning website. The dataset is from the Cleveland database [12]. Not all the personal information of the patients was considered in the dataset due to confidentiality reasons. The dataset consists of 14 features and 1025 patient records. All the attributes that are used to predict the response variable, which is the occurrence of a heart disease, that has a Boolean value.

The second dataset is the Breast Cancer Wisconsin (Diagnostic) Data Set from UCI machine learning repository. The owners of the dataset are William H. Wolberg, Nick Street, and Olvi L. Mangasarian [13]. The data consists of 569 rows where each row represents a record of a patient and the 31 columns that represent features that describe each row. The 9 first attributes are repeated by the mean value (mean), standard error (SE), and mean of the three largest values (worst). The response is the diagnosis for the patients with 1 that represents a malignant tumor, and 0 that represents a benign tumor.

The programming language used is Python and there are many modules to apply parallel computing. In this experiment, the module multiprocessing is used with other important Big Data modules such as Pandas, NumPy, and math [14]. The Spider python platform and the shell command prompt were used in this experiment.

A. KNN algorithm using Multiprocessors

The datasets are fitted to implement KNN model with 10 cross-validation, and to improve the KNN algorithm performance with Big Data sets by the use of python module called multiprocessing that distributes the tasks on the available CPU processors to decrease the execution time. The parallelism has taken place in the *Calculate* function to distribute the data between the specified number of processors and calculate the accuracy for the specified processor to gather the result to print it. The number of processors used were 2, 3, 4,6, and 8 processors.

B. KNN algorithm using Multi-threading

The KNN classifier is used with the multiprocessing library in python to execute the algorithm in multi threads. Each dataset is fitted to implement KNN model with 10 cross-validation and the multi-threading in the computation of the algorithm due to the complexity of computation and takes time in the exaction of the algorithm. The number of threads is 2,4,6,8, and 10 threads with different values of K.

IV. RESULTS

The result for the Heart Test Data Set and Breast Cancer Wisconsin (Diagnostic) Data Set using both the multiprocessing and multi-threading is presented in this section to compare between the different number of processors and the number of threads. Also, to choose the optimal K value with the highest accuracy for each dataset.

A. Heart Test Dataset Result using KNN Algorithm Multiprocessors

Fig. 1. shows the time taken for execution of the KNN algorithm with K value as 3 with different number of processors and it shows that it takes nearly 55 seconds with two processors. The least time required for this dataset is 25 second with the accuracy rate of 87% for K=3.

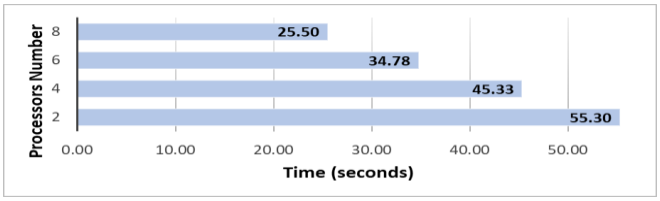

Fig. 1. The result of multiprocessors with K =3.

Moreover, Fig. 2. presents that the shortest time given is 54 seconds with 8 processors, where the time is doubled for 2 processors of KNN algorithm with K value is 5 and the accuracy of heart test dataset is 76.7%, a decreased rate when compared with K=3.

Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.

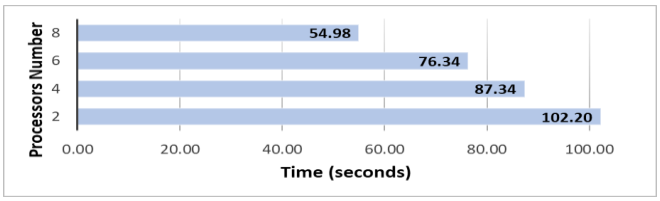

Fig. 2. The result of multiprocessors with K =5.

The decrement of time when K=7 between each trial is low in contrast with time given when K=3 where it had nearly 10 seconds decrement between each trial. However, the last trial with 8 processors achieved the shortest time as illustrated below in Fig. 3. Though, the accuracy reduced when the K value increased to reach 74.3% for K=7.

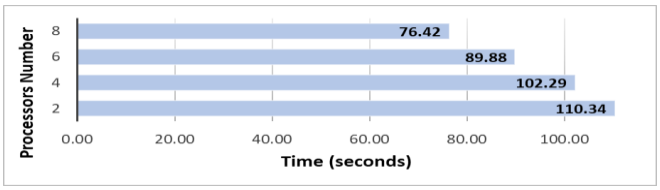

Fig. 3. The result of multiprocessors with K =7.

The execution time with 8 processors was the lowest with 90 seconds and with 2 processors shows the longest time with 154 seconds as shown in Fig. 4. However, the accuracy is 78.6% for K=9 compared with K= 5 and K=7 there is an increase in the accuracy.

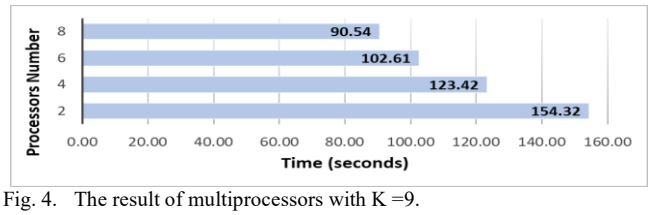

Fig. 4. The result of multiprocessors with K =9.

Comparing between the different number of processors that reflects a good decrement of the average time along with the increment of the processors is displayed in Fig. 5. The time was 120 seconds for the sequential K-NN which was reduced to 105 seconds when using 2 processors. In addition, there is a decrease in the parallel KNN compared with sequential KNN to reach about half time required to execute the algorithm as shown in figure. The best K value for the heart test dataset is 3 with 87.0% accuracy.

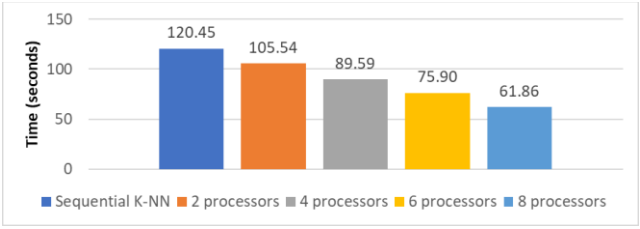

Fig. 5. Comparison between different processors using average time.

The speedup is calculated with the Equation (1) it is the time required for sequential KNN divided by the parallel time as shown in TABLE I. As the increase of the number of the processors the speedup increases to reach 1.95.

$$S = T_S / T_P \quad (1)$$

TABLE I. THE RELATIVE SPEEDUP OF THE PROPOSED PARALLEL CLASSIFIER.

Number of processors	*Speedup*
2 processors	1.14
4 processors	1.34
6 processors	1.59
8 processors	1.95

Also, the efficiency is computed using equation (2) where the speedup is divided by the number of processors. TABLE II shows the efficiency and its decline as the number of the processors increment to reach 0.24 with 8 processors.

$$E = S/P \quad (2)$$

TABLE II. THE EFFICIENCY OF THE PROPOSED PARALLEL CLASSIFIER.

Number of processors	***Efficiency***
2 processors	0.57
4 processors	0.34
6 processors	0.26
8 processors	0.24

B. Heart Test Dataset Result using KNN Algorithm Multi-threading

In this part, multi-threading is applied on the KNN algorithm with the heart patient's dataset with different number of threads and K values. As shown in Fig. 6, the execution of multi-threading is slower with 2 threads and astronomically fast with 10 threads. The K value is 3 and the accuracy is 87.0%.

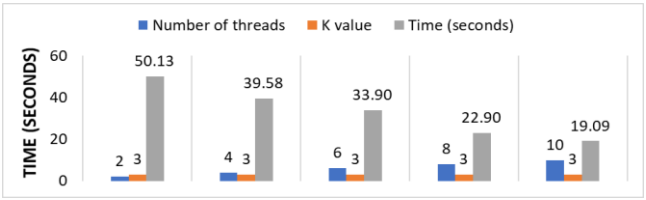

Fig. 6. The result of multi-threading with K =3.

The value of K=5 is presented in Fig. 7 where the time of execution decreased with the increment of the number of threads to reach the least time required with 10 threads. Nevertheless, the accuracy decreased to achieve 76.7% compared with K=3.

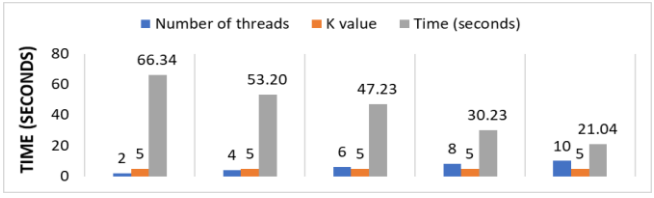

Fig. 7. The result of multi-threading with K =5.

In Fig. 8 the K value=7 with 76.7% accuracy. The best time required to run is using 10 threads with 23 seconds and as the increase of the threads the time decreased. But, the growth of K value leads to more time required for computation and the execution of the threads.

Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.

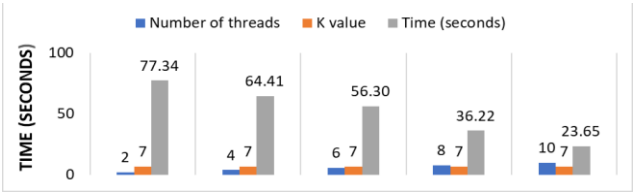

Fig. 8. The result of multi-threading with K =7.

The time taken to execute the KNN algorithm with K value as 9 is too slow with 120 seconds using 2 threads, but it is faster with 32 seconds using 10 threads as demonstrated in Fig. 9. The accuracy was 78.6% with K=9.

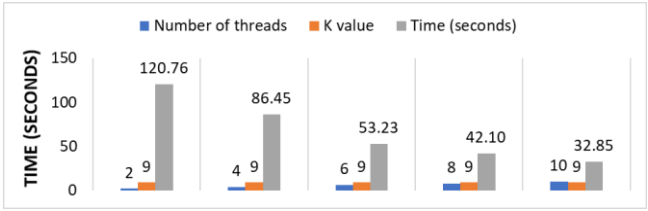

Fig. 9. The result of multi-threading with K =9.

There are several numbers of threads demonstrates in Fig. 10 that shows the average time required using multi-threading for KNN algorithm. The time accelerates with a decline of the number of threads. The best number of threads used for this dataset is 10 threads with the least time required to perform the KNN algorithm.

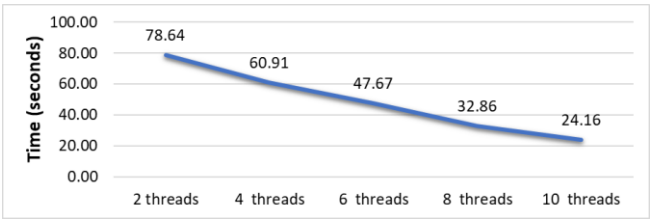

Fig. 10. Comparison between various threads with the average time.

C. *Breast Cancer Wisconsin (Diagnostic) Data Set Result using KNN Algorithm Multiprocessors*

The Breast Cancer Wisconsin (Diagnostic) Data Set is fitted to KNN model with multiprocessors for different number of K as illustrated in Fig. 11 that increasing the processors decreases the time to reach 35 seconds with 87.4%. Both 6 and 8 processors execute the algorithm with less time required.

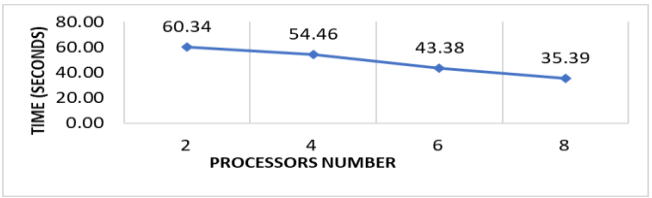

Fig. 11. The result of multiprocessors with K =3.

The K value increased to extend to 5 with the reduction of the time using 8 processors as displayed in Fig. 12. The accuracy using K= 5 is 87.0% with the least time required to reach 82 seconds. In 2 processors the time required is 110 seconds, but using 8 processors the time required is 82 seconds.

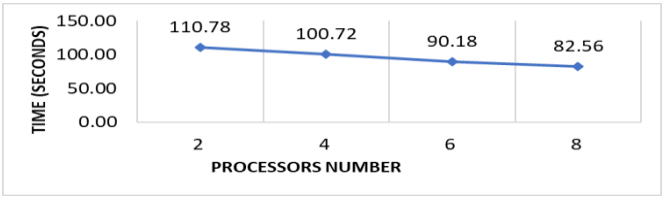

Fig. 12. The result of multiprocessors with K =5.

Fig. 13 illustrates the result with K=7 and the accuracy is 87.0% using multiprocessors. The best number of processors to use is 8 with 90 seconds. As the number of processors decrease the time increases and the most effective number of processors is 8 processors.

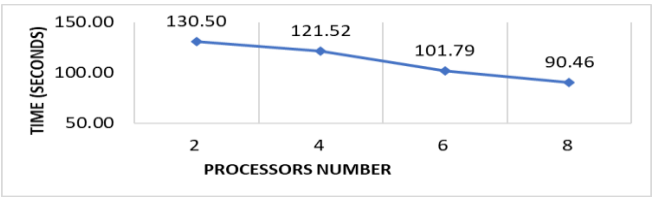

Fig. 13. The result of multiprocessors with K =7.

The K value is equal to 9 with a different number of processors represented in Fig. 14 to achieve 87.0% in the accuracy. The time using 2 processors is 166 seconds but using 8 processors had reduced the time to be 121 seconds. Moreover, with the growth of K the time required for the computation increase as shown if the figure.

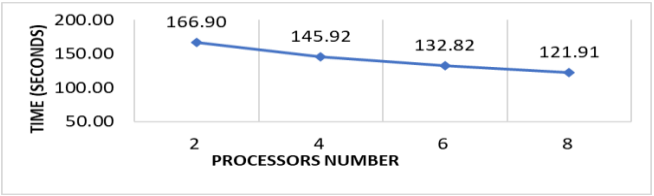

Fig. 14. The result of multiprocessors with K =9.

Fig. 15 represents the different number of processors with the lowest average time is using 8 processors in 82 seconds compared with 2 processors in 117 seconds. Moreover, there is a decline in the time comparing the sequential KNN with the parallel KNN. The highest accuracy is using k=3 with 87.4% and using 8 processors is suitable for this dataset to predict the type of tumor.

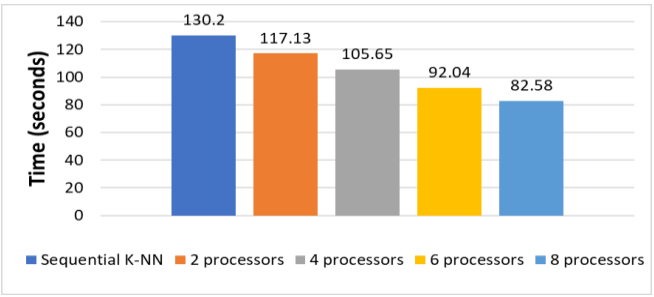

Fig. 15. Comparison between different processors using average time.

The speedup is calculated for the second dataset to evaluate the KNN algorithm. The speedup escalates to reach 1.58 with 8 processors as displayed in TABLE III. as the number of the processors rise the speedup approximately rise by 0.02.

Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.

TABLE III. THE RELATIVE SPEEDUP OF THE PROPOSED PARALLEL CLASSIFIER.

Number of processors	*Speedup*
2 processors	1.11
4 processors	1.23
6 processors	1.41
8 processors	1.58

The efficiency is assessed for the second dataset as well to check the KNN parallel performance. As demonstrated in TABLE IV. the efficiency drops with a greater number of the processors to 0.20 using 8 processors. The efficiency using 2 processors was 0.56 but shrunk with more processors.

TABLE IV. THE EFFICIENCY OF THE PROPOSED PARALLEL CLASSIFIER.

Number of processors	***Efficiency***
2 processors	0.56
4 processors	0.31
6 processors	0.24
8 processors	0.20

D. Breast Cancer Wisconsin (Diagnostic) Dataset Result using KNN Algorithm with Multi-threading

In Fig. 16 the result using 10 cross validation for several numbers of threads with K=3 and the accuracy is 87.4%. Increasing the number of threads leads to decrease the time for execution the algorithm with 10 threads the lowest time required to reach 11 seconds.

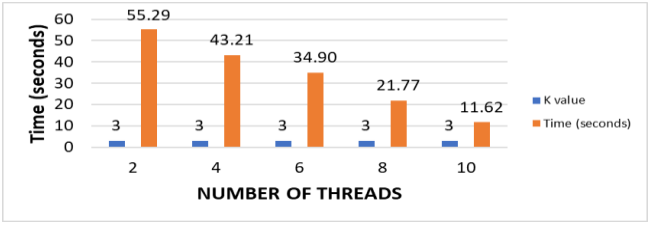

Fig. 16. The result of multi-threading with K =3.

Fig. 17 shows that increasing K=5, the time increased and the accuracy decreased to 87.0%. The figure shows the decline in time by increasing the thread number to reach 10 threads with 16 seconds.

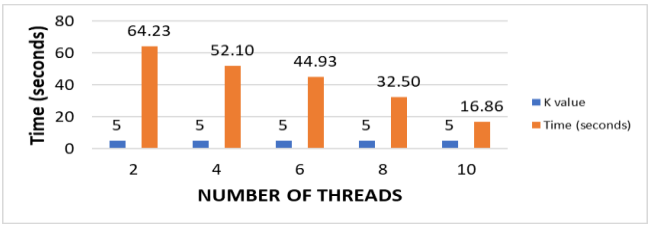

Fig. 17. The result of multi-threading with K =5.

As presented in Fig. 18, the result using K=7 and the accuracy is 87.0%. The escalation of the threads shows a reduction in the time for applying the algorithm with 10 threads the fastest time by 26 seconds.

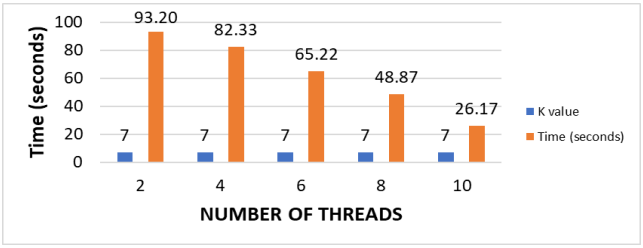

Fig. 18. The result of multi-threading with K =7.

The K value is 9 by multi-threading with the accuracy of 87.0%. The more threads used, the less time for execution. Using 2 threads, the time reached 154 seconds, and when using 10 threads, it reached 30 seconds as shown in Fig. 19. The use of 10 threads helped in speeding the algorithm with the Breast Cancer Wisconsin (Diagnostic) Data Set.

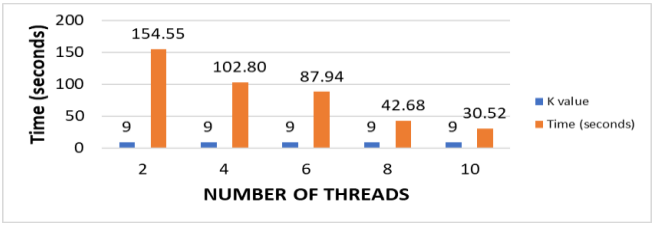

Fig. 19. The result of multi-threading with K =9.

Fig. 20 compares between various thread number with the average execution time. The higher the number of the threads, the less time taken to execute the KNN algorithm. The 10 threads are the best number of threads with 21 seconds and the best K value for this dataset is using 3 with the accuracy 87.4%.

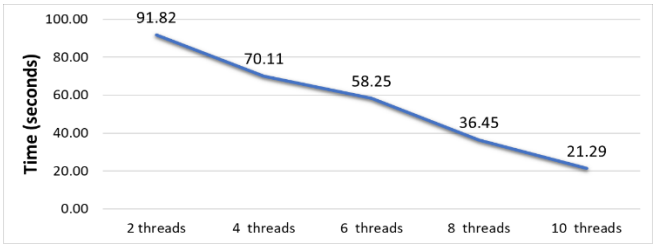

Fig. 20. Comparison between various threads with the average time.

V. CONCLUSION

In conclusion, the KNN is executed using multiprocessors and multi threads to speed up the KNN algorithm. The KNN algorithm is used to compare the parallel K-NN with the sequential version with the time. The results show that the sequential algorithm required more time indicating that the parallel KNN outperformed the sequential. The speedup shows that the greater number of processors, the more the speedup and comparing the time for different number of processors resulted in less time required to execute the algorithm with the two datasets for higher number of processors. Moreover, the use of 2,4,6,8, and 10 threads showed the decrease in time to calculate the KNN algorithm with the increasing number of threads for both the datasets.

Some of the limitation with the python libraries for parallel programming are new with less reference to gain the knowledge and less application performed with python compared with the other programming languages. Also, the experiments of the algorithm were performed on a single PC,

Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.

which resulted in a several crashes of the system and the difficulty to manage the timing for the proposed algorithm. Future work includes using the cloud to execute the parallel algorithm and use more datasets to perform the KNN in parallel.

REFERENCES

[1] M. S. Nobile, P. Cazzaniga and A. T. a. D. Besozzi, "Graphics processing units in bioinformatics,computational biology and systems biology," *Briefings in Bioinformatics,* pp. 1-16, 2016.

[2] S. B. Imandoust and M. Bolandrafta, "Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events: Theoretical Background," *Journal of Engineering Research and Applications,* vol. 3, no. 5, pp. 605-610, 2013.

[3] S. Dhanabal and S. Chandramathi, "A Review of various k-Nearest Neighbor Query Processing Techniques," *International Journal of Computer Applications,* vol. 31, no. 7, pp. 14-22, 2011.

[4] C. Zhang, F. Li and J. Jestes, "Efficient Parallel kNN Joins for Large Data in MapReduce," in *ACM International Conference Proceeding Series*, 2012.

[5] N. Sismanis, N. Pitsianis and X. Sun, "Parallel search of k-nearest neighbors with synchronous operations," in *IEEE Conference on High Performance Extreme Computing*, 2012.

[6] N. Rajani, R. McArdle and I. Dhillon, "Parallel k nearest neighbor graph construction using tree-based data structures," 2015.

[7] Y. Pu, J. Peng, L. Huang and J. Chen, "An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL," in *IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015.

[8] M. A. Patwary , N. . R. Satish , N. Sundaram , J. Liu , P. Sadowski , E. Racah , S. Byna , C. Tull , W. Bhimji , Prabhat and P. Dubey, "PANDA: Extreme Scale Parallel K-Nearest Neighbor on Distributed Architectures," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Chicago, 2016.

[9] M.-W. Cao, L. Li, W.-J. Xie , W. Jia , Z.-H. Lv , L.-P. Zheng and X.-P. Liu, "Parallel K Nearest Neighbor Matching for 3D Reconstruction," *IEEE Access,* vol. 7, pp. 55248-55260, 2019.

[10] Y. Pu, X. Zhao, G. Chi, S. Zhao, J. Wang, Z. Jin and J. Yin, "Design and implementation of a parallel geographically weighted k-nearest neighbor classifier," *Computers & Geosciences,* pp. 111-122, 2019.

[11] J. Park and D. H. Lee, "Parallelly Running k-Nearest Neighbor Classification Over Semantically Secure Encrypted Data in Outsourced Environments," *IEEE Access,* vol. 8, pp. 64617-64633, 2020.

[12] " UCI Machine Learning Repository," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/heart+disease. [Accessed 12 9 2020].

[13] "UCI Machine Learning Repository," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic). [Accessed 12 9 2020].

Authorized licensed use limited to: Carleton University. Downloaded on November 30,2022 at 19:06:06 UTC from IEEE Xplore. Restrictions apply.