2022 14th International Conference on Computational Intelligence and Communication Networks (CICN) | 978-1-6654-8771-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CICN56167.2022.10008343

Cloud-Based Reconfigurable Hardware Accelerator for the KNN Classification Algorithm

Awos Kanan
Dept. of Computer Engineering
Princess Sumaya University for Technology
Amman 11941, Jordan
a.kanan@psut.edu.jo

Amal Taha
Dept. of Computer Engineering
Princess Sumaya University for Technology
Amman 11941, Jordan
ama20180264@std.psut.edu.jo

***Abstract*—The K-Nearest Neighbor algorithm is a supervised machine learning algorithm that is used for classification problems. The execution time of this algorithm could be extremely high, especially for huge and high-dimensional datasets. The objective of this work is to design and implement efficient parallel hardware architectures to accelerate the KNN classifier. The proposed architectures are implemented using FPGA on Intel DevCloud. Experimental results show that the proposed hardware implementation of the algorithm is 10.7 times faster than the software implementation with 96.6% classification accuracy for a benchmark classification dataset.**

***Index Terms*—KNN Classifier, OpenAPI, DevCloud, DPC++, Hardware Acceleration, FPGAs, Image Classsification.**

I. Introduction

Data mining is concerned with finding useful information from large datasets. Among the high-level tasks that data mining typically involves are classification, regression, clustering, summarizing, dependency modeling, and detecting changes and deviations [1], [2]. The K-Nearest Neighbor (KNN) classification algorithm [3] is one of the most popular algorithms that is widely used in many machine learning and data mining applications. Despite the simplicity of this algorithm, it could be heavy for the hardware when used for classifying huge and high-dimensional datasets [4]. The classification model of the KNN algorithm involves two main kernels that are computationally intensive; the distance calculation unit, and the sorting unit. Enhancing the execution time of these kernels is of great interest to enhance the overall performance of the KNN algorithm.

Hardware acceleration is the process of using specialized hardware architectures to execute computationally-intensive tasks to speed up the overall execution time of the task to be accelerated. Typically, Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), and Application Specific Integrated Circuits (ASICs) are widely used for this purpose with GPUs being dominant in this field. FPGAs are more flexible and energy-efficient compared to GPUs while being much cheaper than ASICs. As a result of the increasing demand on high performance computing using FPGAs, new generations of hardware design platforms using High-Level Synthesis (HLS) tools are being introduced to automatically synthesize high-level descriptions of digital systems [5], [6].

Several FPGA-based hardware accelerators have been proposed for the KNN classifier. In [7], the authors implemented the KNN algorithm using reconfigurable hardware architecture for text classification with a speedup of 15 over a single CPU and 1.5 over a multi-threaded CPU. Another hardware architecture to accelerate the KNN classifier on mobile devices has been proposed in [8] that is 127 times faster than its software counter-part. The authors in [9] proposed a Software Defined Networking architecture to secure smart homes using the KNN classifier for malicious traffic detection and the parallel processing power of FPGAs.

Recent advances in hardware acceleration services provided on the cloud result in more interest in cloud-based FPGA acceleration of machine learning and data mining algorithms including the KNN classifier [10]. The main advantage of using FPGAs on the cloud, compared to the aforementioned classical approaches, is the unlimited access to huge capabilities, including high memory bandwidths, for implementing and testing hardware acceleration architectures. The aim of this paper is to accelerate the KNN classification algorithm using the recent high-level language Data Parallel C++ (DPC++) that is used for cross-architecture and heterogeneous hardware implementations [11]. Detailed hardware implementations of the main computational kernels of the algorithm along with experiemntal results and comparisons with software implementations, all on the cloud, are provided.

The remaining of this paper is organized as follows; Section II provides background information on the KNN classification algorithm and the used acceleration platform. The proposed architectures along with implementation methodologies are discussed in Section III. Comparisons and experimental results are presented in Section IV. Finally, Section V concludes the paper.

II. Background

A brief background on the KNN classification algorithm and the cloud-based implementation platform and tools used to accelerate this algorithm are discussed in the following subsections.

978-1-6654-8771-9/22/$31.00 ©2022 IEEE
DOI: 10.1109/cicn.2022.53

A. K-Nearest Neighbor Algorithm

The KNN classification algorithm is an efficient and easy to implement classifier that relies on similarity measures [12] such as Manhattan, Euclidean, and Cosine distances to classify data samples of different machine learning datasets. The Euclidean distance, for instance,is the straight-line segment between two points in a mutli-dimensional coordinates space. The algorithm observes the whole dataset differences and then takes the K closest of them to the test data.

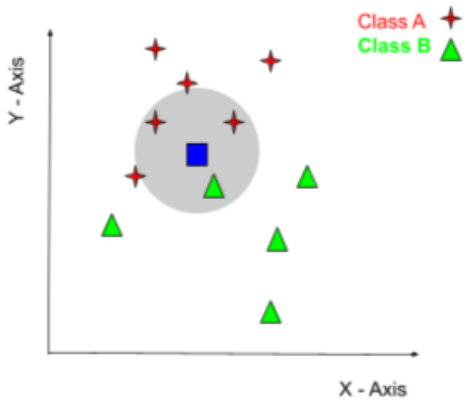

Fig. 1. Example KNN Classification with two classes A and B.

In Fig. 1, we have two classes A and B that are considered training data since they are already labeled. The blue square represents an unclassified, or test, data that we need to measure its similarity to either class A or B. We can find that $K = 3$ gives us a circle that covers two elements from class A and one from B, thus we predict that the tested entry will be classified as class A. Several approaches can be used to select the value of K [13]. If the range of values to select from is not big, we can try every possible value and select the one that maximizes the classification accuracy.

B. Intel OneAPI DPC++ Platform

Intel DevCloud is a modern innovation sandbox that provides powerful tools for developers to program cross-architecture applications on different platforms they provide on the cloud such as; OpenVino, HLS tools, OpenCl, RTL, and OneAPI. One of the most interesting advantages of using Intel DevCloud is that it has free trial with unlimited access to nodes with CPUs, GPUs, and FPGAs. Intel DevCloud is a highly scalable sandbox that is easy to learn and use. The new DPC++ extension is one of the powerful tools Intel DevCloud provides to design, implement, and analyze an FPGA-based solution. Developers can login to the cloud through direct SSH connection or through a Jupyter Notebook that Intel provides online for application development and acceleration [11].

OneAPI is a platform with several toolkits that are provided by Intel as part of DevCloud. DPC++ is one of these toolkits. It is a high-level cross-architecture language that is inherited from C++ and uses SYCL 2020 extensions with more advanced features. SYCL 2020 is another high-level language that is used to improve the design of multiple heterogeneous-systems. The Intel OneAPI DPC++/C++ compiler maps the instructions of high-level code into hardware operations and controls this mapping to utilize available hardware resources efficiently [14].

III. Methodologies

A. Memory and Device Management in DPC++

To start using DPC++ for FPGA-based acceleration, we need to include some necessary header files, select the device that will be used, and submit it to the job queue. Two main ways are available for memory management while offloading the computations using DPC++; Unified Shared Memory, and Buffers and Accessors. Unified Shared Memory is a pointer-based method that uses a shared memory between the host and the kernel such that both devices use the same data objects. Buffers and Accessors, on the other hand, is a new powerful way of interacting with the device and handling data dependencies. It uses SYCL extensions to create a buffer and an accessor for each object where the buffer holds the data to be managed while accessors have control over the buffer objects. This Method is the one that is used to describe the proposed architectures in this work.

There are four main methods that are used to invoke the kernel (FPGA device):

- **Single Task**: Perform one task on the device kernel with no parallelism.
- **Parallel Kernel**: Invoke the kernel N times at the same time to allow for parallelism on the device. Not necessarily all N operations will be executed at the same time.
- **ND-Range Kernel**: This method uses the same syntax of Parallel Kernel. However, it allows for better performance by mapping executions to hardware resources. This happens by grouping each execution into a different compute unit on the kernel to achieve occupancy and parallelism.
- **Hierarchical Kernel**: This method has the same idea and functionality of the ND-Range except that it is represented in more hierarchical way for better description of the code.

B. The Proposed Accelerator

High-Level acceleration Platforms, including DPC++, allow developers to write high-level programs that run on a host computer with the capability to offload slow portions of the code to some acceleration devices such as GPUs and FPGAs. Fig. 2 shows a high-level design of the proposed FPGA-based accelerator for the KNN classification algorithm.

The main advantage of the used acceleration approach lies in abstracting the code and making it easier to interact with the FPGA without the need to explicitly define each pin and module the same way in Hardware Description Languages (HDLs), thus implementing more complex designs with less time and efforts with the ability to easily debug and separate host and kernel codes.

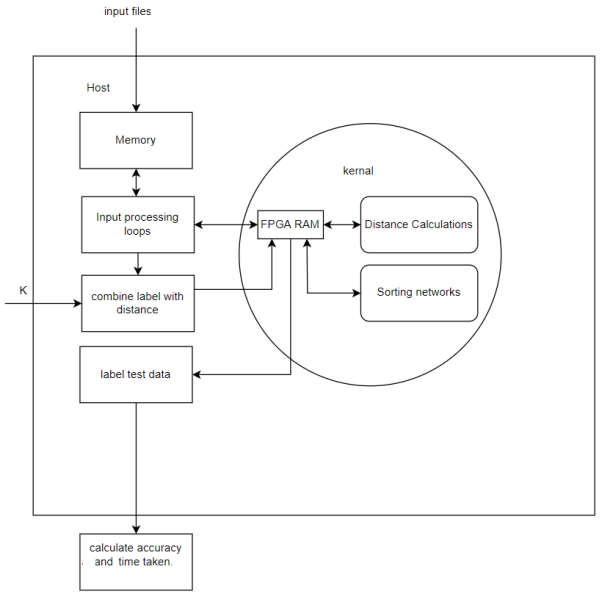

Fig. 2. Block Diagram of the Proposed KNN Architecture.

At first, we have a dataset with training and testing data that are read and saved in vectors inside the heap memory of the host device so that we do not overwhelm the FPGA with massive amount of reading and writing operations. The vectors are sent to the kernel memory through buffers and accessors that allow us to separate host and kernel RAMs. The most critical and time consuming task is data transfer to the kernel. The proposed acclerator has two main units inside the kernel; the similarity distance unit and the sorting unit.

A dataset that consists of a large number of samples with many features can be implemented using vectors or 2D arrays. Since it is more efficient to deal with vectors in DPC++, we decided to convert input datasets into 1D vectors as show in Fig. 3.

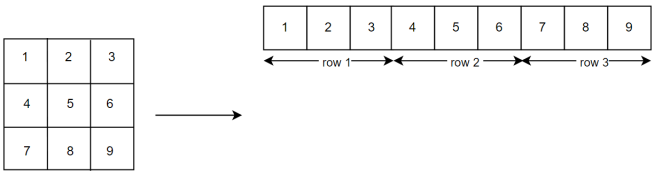

Fig. 3. Converting a 2D array to 1D vector.

As shown if Fig. 4, multiple training data samples are fed to the distance calculation unit in parallel. The distance between training sample cells and the current test sample is then calculated. This unit is implemented using a parallel for loop at the aim of performing as much of the computations in the loop iterations in parallel as possible. What happens inside the kernel depends on the hardware specifications of the FPGA. Hence, it is very important to refer to analysis reports of the hardware image to analyze the performance of the accelerator to decide what to do to achieve better performance. For Euclidean distance, the loop iterates through all features of data samples, as they represent the dimensions for the KNN algorithm. We add the differences on each dimension and multiply them with their value to get their square. The result of these calculations are stored inside the initial vector that's submitted to the buffer as an input. The result vector is the RAM that stores the resulted distance of each of the entries and deliver it back to the original vector that was submitted to the kernel with write accessor permission. The proposed distance calculation unit supports two more similarity measures; Manhattan distance and Cosine distance.

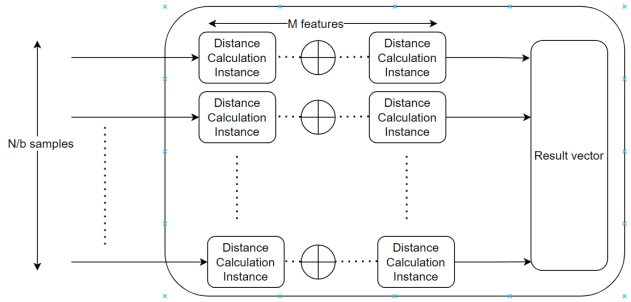

Fig. 4. Parallel distance calculation unit.

The KNN algorithm relies on sorting operations to find the nearest neighbors. Sorting Networks is a common principle in designing sorting hardware algorithms. One of them is the Merge Sort algorithm [15] that performs sorting by splitting the data into powers of 2 packet sizes and keeps doing this until the packet size is 2 then swaps the maximum element with the minimum element if the maximum takes the smaller index. When the packet size is larger, it sorts it in an increasing order and keeps doing this in parallel for the packets until the whole list is sorted. Merge Sort Sequence is a way to merge two sorted lists into one that is sorted too, as shown in the example in Fig. 5. The proposed accelerator relies on using parallel for constructs to implement the sorting operation on the resulting vectors in parallel.

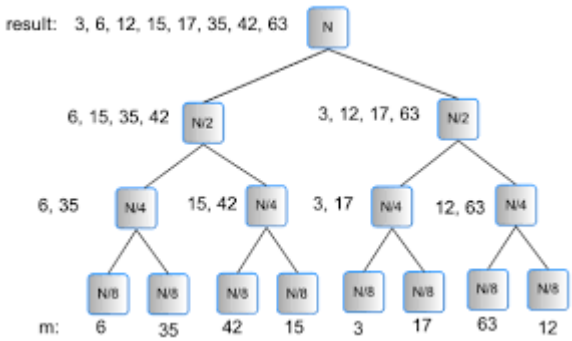

Fig. 5. Merge Sort Example [16].

IV. Experimental Results

The complete KNN algorithm is implemented using DPC++ high-level language, targeting Arria 10 GX FPGA device, and

compared to a Python implementation on the same cloud platform (Intel DevCloud). As discussed in the previous section, the complete KNN algorithm is implemented in a way so that some parts of the code are executed on the host while offloading the remaining computationally-intensive parts to the FPGA kernel.

Results reported in this section are based on a OneAPI dataset that is provided with the platform documentation for testing purposes. The dataset has 16,000 training samples and 4,000 test samples, five features all represented as real numbers larger than 0, and five labels (from 0 to 4) [17].

In addition to comparing the complete DPC++ and Python implementations of the KNN algorithm, we provide a detailed comparisons for the main kernel units; the distance calculation unit and the sorting unit. Table I shows the execution times of both FPGA and Python implementations of the distance calculation unit with three different similarity measures (Euclidean, Manhattan, and Cosine distances), the sorting unit, and the complete KNN algorithm.

TABLE I
EXECUTION TIMES FOR THE FPGA AND PYTHON IMPLEMENTATIONS.

Unit	Execution Time (ms)	
	Proposed	Python
Distance - Euclidean	0.83	70
Distance - Manhattan	0.97	76
Distance - Cosine	2	125
Sorting	231	28
Complete KNN	31,058	334,279

From the above table, it is obvious that the proposed FPGA implementations of the distance calculation unit perform the required computations in much less time compared to the Python implementations for all similarity measures. Fig. 6 shows the speedup achieved for each of the three measures. The lowest speedup is achieved with the Cosine distance due to the required more complex operations, including division, compared to the other two measures, and the accompanying overhead of creating buffers and accessors.

The speedup achieved for the proposed sorting unit is around 8x, which is much less than the speedups achieved for the proposed distance calculation units. This is due to data dependencies in the proposed sorting unit that limits the degree of parallelization of the loops that are offloaded to the kernel.

The complete implementations of the algorithm are based on the recommended value of $K = 126$, which is the square root of the dataset size [13]. The overall speedup of the proposed implementation over the Python implementation is 10.7x with classification accuracy of 96.675% for the benchmark dataset described earlier in this section.

The utilization of the FPGA resources occupied by the complete implementation of the KNN algorithm are shown in Table II. As shown in this table, the proposed implementation uses small portions of the available resources, which makes it

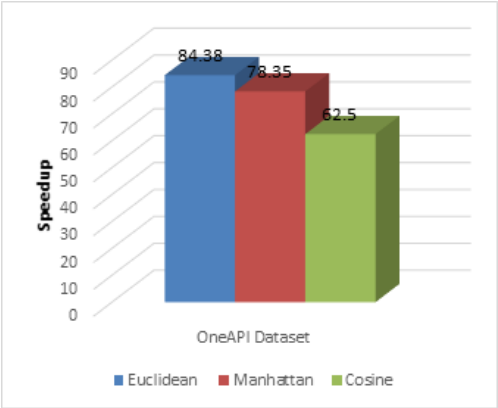

Fig. 6. Speedup for different similarity measures.

area and power efficient while being scalable to higher degrees of parallelism and larger datasets.

TABLE II
UTILIZATION OF FPGA RESOURCES.

Resource	Utilization
LookUp Tables (LUTs)	49752 (6%)
Flip flops (FFs)	60654 (4%)
Block RAMs	292 (11%)
Digital Signal Processing (DSPs)	19 (1%)

V. CONCLUSION

The work presented in this paper focuses on using the cloud-based development platform DevCloud OneAPI, provided by Intel, to accelerate the KNN classification algorithm using FPGAs. The emerging DPC++ high-level language is used to implement the complete algorithm. The proposed implementation achieves better performance compared to a Python software implementation, on the same cloud platform, with high classification accuracy using small portions of the available hardware resources.

REFERENCES

[1] S.-H. Liao, P.-H. Chu, and P.-Y. Hsiao, "Data mining techniques and applications–a decade review from 2000 to 2011," *Expert systems with applications*, vol. 39, no. 12, pp. 11 303–11 311, 2012.

[2] J. F. Pinto da Costa and M. Cabral, "Statistical methods with applications in data mining: A review of the most recent works," *Mathematics*, vol. 10, no. 6, p. 993, 2022.

[3] K. Taunk, S. De, S. Verma, and A. Swetapadma, "A brief review of nearest neighbor algorithm for learning and classification," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. IEEE, 2019, pp. 1255–1260.

[4] J. Masek, R. Burget, J. Karasek, V. Uher, and M. K. Dutta, "Multi-gpu implementation of k-nearest neighbor algorithm," in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2015, pp. 764–767.

[5] A. A. Al-Aghbari and M. E. Elrabaa, "Cloud-based fpga custom computing machines for streaming applications," *Ieee Access*, vol. 7, pp. 38 009–38 019, 2019.

[6] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing," in *2016 26th International conference on field programmable logic and applications (FPL)*. IEEE, 2016, pp. 1–10.

[7] K. R. Townsend, S. Sun, T. Johnson, O. G. Attia, P. H. Jones, and J. Zambreno, "k-nn text classification using an fpga-based sparse matrix vector multiplication accelerator," in *2015 IEEE International Conference on Electro/Information Technology (EIT)*. IEEE, 2015, pp. 257–263.
[8] M. A. Mohsin and D. G. Perera, "An fpga-based hardware accelerator for k-nearest neighbor classification for machine learning on mobile devices," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2018, pp. 1–7.
[9] H. Gordon, C. Park, B. Tushir, Y. Liu, and B. Dezfouli, "An efficient sdn architecture for smart home security accelerated by fpga," in *2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2021, pp. 1–3.
[10] A. Lu, Z. Fang, N. Farahpour, and L. Shannon, "Chip-knn: A configurable and high-performance k-nearest neighbors accelerator on cloud fpgas," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 139–147.
[11] J. Fuentes, D. López, and S. González, "Teaching heterogeneous computing using dpc++," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 354–360.
[12] S. Santini and R. Jain, "Similarity measures," *IEEE Transactions on pattern analysis and machine Intelligence*, vol. 21, no. 9, pp. 871–883, 1999.
[13] S. Zhang, X. Li, M. Zong, X. Zhu, and D. Cheng, "Learning k for knn classification," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 3, pp. 1–19, 2017.
[14] R. Nozal and J. Bosque, "Straightforward heterogeneous computing with the oneapi coexecutor runtime. electronics 2021, 10, 2386," 2021.
[15] U. A. Korat, P. Yadav, and H. Shah, "An efficient hardware implementation of vector-based odd-even merge sorting," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. IEEE, 2017, pp. 654–657.
[16] "Merge sort," (Date last accessed 10-October-2022). [Online]. Available: http://selkie.macalester.edu/csinparallel
[17] "Oneapi dataset," (Date last accessed 10-October-2022). [Online]. Available: https://docs.oneapi.io/versions/latest/onedal/examples/cpp