electronics

MDPI

Article

A Modified KNN Algorithm for High-Performance Computing on FPGA of Real-Time m-QAM Demodulators

David Marquez-Viloria [1,*], Luis Castano-Londono [1] and Neil Guerrero-Gonzalez [2]

[1] Faculty of Engineering, Insituto Tecnologico Metropolitano, Calle 73 No. 76A-354 Via al Volador, Medellin 050034, Colombia; luiscastano@itm.edu.co
[2] Department of Electrical, Electronic and Computer Engineering, Universidad Nacional de Colombia, Carrera 27 No. 64-60, Manizales 170004, Colombia; nguerrerog@unal.edu.co
* Correspondence: davidmarquez@itm.edu.co; Tel.: +57-4-440-51-00

Abstract: A methodology for scalable and concurrent real-time implementation of highly recurrent algorithms is presented and experimentally validated using the AWS-FPGA. This paper presents a parallel implementation of a KNN algorithm focused on the m-QAM demodulators using high-level synthesis for fast prototyping, parameterization, and scalability of the design. The proposed design shows the successful implementation of the KNN algorithm for interchannel interference mitigation in a 3 × 16 Gbaud 16-QAM Nyquist WDM system. Additionally, we present a modified version of the KNN algorithm in which comparisons among data symbols are reduced by identifying the closest neighbor using the rule of the 8-connected clusters used for image processing. Real-time implementation of the modified KNN on a Xilinx Virtex UltraScale+ VU9P AWS-FPGA board was compared with the results obtained in previous work using the same data from the same experimental setup but offline DSP using Matlab. The results show that the difference is negligible below FEC limit. Additionally, the modified KNN shows a reduction of operations from 43 percent to 75 percent, depending on the symbol's position in the constellation, achieving a reduction 47.25% reduction in total computational time for 100 K input symbols processed on 20 parallel cores compared to the KNN algorithm.

Keywords: constellation diagram; FPGA; K-nearest neighbors; machine learning; optical communications; QAM

Citation: Marquez-Viloria, D.; Castano-Londono, L.; Guerrero-Gonzalez, N. A Modified KNN Algorithm for High-Performance Computing on FPGA of Real-Time m-QAM Demodulators. *Electronics* **2021**, *10*, 627. https://doi.org/10.3390/electronics10050627

Academic Editor: Yunho Jung

Received: 2 February 2021
Accepted: 28 February 2021
Published: 9 March 2021

1. Introduction

How to implant intelligence inside electronic subsystems is becoming essential in upcoming reconfigurable optical coherent architectures above Terabit transmissions [1]. Recent reports on machine learning applications in coherent systems have described the high recurrency and the complex management of electronic memory based on the minimization of objective functions. However, lately published literature is focused on offline processing, leaving space for research in real-time implementation of machine learning algorithms satisfying requirements of latency and scalability while keeping low levels of power consumption [2,3].

The constellation diagram provides relevant information such as modulation format, optical signal-to-noise ratio (OSNR), vector magnitude error (EVM), and transmission impairments [4]. The need to mitigate transmission impairments has prompted the use of machine learning (ML) techniques on the constellation diagram [5]. Applications cover a variety of techniques such as maximum-likelihood for phase rotation detection [6], nonlinear phase noise mitigation through techniques such as support vector machine (SVM) [7] or K-nearest neighbors (KNN) [8], and intelligent constellation analysis using convolution neural network (CNN)-based deep learning technique [4]. A 2D-plane represents the constellation diagram being possible to apply image processing methods to identify distorted patterns [9–12].

The KNN algorithm is a supervised method widely used in different applications because of its simplicity and effectiveness. The state-of-the-art presents the potential of KNN for distortion mitigation [13–15]. However, these are usually offline implementation, which does not include real applications. Therefore, proposals must include a real-time implementation of the KNN algorithm. For example, in m-QAM schemes, each received symbol must be assigned to a cluster corresponding to the mapping. The operations performed are independent for each symbol from a channel or multiple channels. Thus, it is possible to perform the operations in parallel, making the field-programmable gate array (FPGA) a viable candidate for execution due to the inherent parallelism of these devices.

Central processing units (CPUs) provide high performance in sequential computing but depend on increased clock speed to improve their performance. Even including multicore CPUs, they are not considered to be highly parallelizable devices, which is why in many digital signal processing and machine learning applications, they are being replaced with graphics processing units (GPUs) or FPGAs [16,17]. In optical communications systems, it is necessary to sample from the detectors at high speeds and store them in local memory for processing. It implies the use of high-speed digital interfaces such as those defined in the JESD204B standard. These interfaces are currently available in Application-Specific Integrated circuit (ASIC) and FPGAs but are not available for CPUs and GPUs. Therefore, ASICs are the primary choice for digital signal processing in optical communications due to their high performance and energy efficiency. However, they have a high manufacturing cost, low flexibility, and a long development cycle [18]. It has led to FPGA-based devices currently being the norm in optical communications electronics at the development level due to the direct connection through high-speed interfaces, the development of parallel architectures for DSP algorithms, and the possibility of converting the designs into ASICs for commercial applications [10,19].

KNN accelerators' implementation using high-level synthesis (HLS) tools has been presented with OpenCL in [20] and Vivado HLS in [21]. Although these tools allow accelerating the design flow, there are still aspects related to the use of optimization directives and design space evaluation that require considerable effort. In some cases, the need for a certain level of hardware expertise to reach an adequate level of optimization persists [22]. For this reason, some authors have focused on developing tools to simplify the use of optimization directives as in [22] or give guidelines to understand the operation and performance effects of some types of algorithms when using these directives as in [23]. There are several FPGA real-time implementations of the KNN algorithm in the literature. In Reference [24] the authors refer to different works on KNN accelerators targeting FPGA and System-on-Chips (SoCs). They report as the main shortcomings of these works that they are not general-purpose, perform classification of only one sample at a time, work with custom fixed-point numerical representation, and require considerable design changes when there are changes in the parameters or the dataset. Works focused on the implementation of general-purpose KNN accelerators can be found in [20,21,25].

The design proposed in this work is focused on the m-QAM demodulator, uses high-level synthesis for fast prototyping and scalability for quick migration to different FPGA targets. Additionally, we used optimization directives for the design of processing units for maximum concurrency, and multicores incorporated in the design for the parallel processing of symbols from one or multiple channels. The design uses multiple cores of custom processing units for the parallel processing of input symbols. Furthermore, the proposed methodology allows scaling the design by modifying parameters such as the number of cores, the length of training data, the number of input symbols, the number of neighbors, the number of channels, and the m-QAM scheme.

Also, we present a modification of the KNN algorithm to reduce the operations in m-QAM demodulators. To the best of our knowledge, this is a new approach that reduces the number of comparisons needed in the KNN algorithm by using only the training data associated with the 8-connected neighbors containing the label of the input symbol. The

multicore scalable hardware architecture mentioned above was adapted to implement the modified KNN algorithm.

Finally, we validated the architecture using experimental data from a Nyquist-WDM gridless system affected by inter-channel interference (ICI). In this paper, we study the effect of ICI in a multicarrier optical transmission manifested in the constellation diagram as blurred data-symbols (or clusters) and intersecting each other. One of the objectives of this paper consists of associating each distorted data-symbol to the proper cluster despite the blurring, based on the estimation of belonging functions according to analysis of two-dimensional (2D) Euclidean distances following the principles of the k-nearest neighbor (KNN) algorithm. We validate the results comparing the real-time FPGA implementation and the MATLAB offline execution. Also, we verified our operations reduction technique comparing our 20 parallel cores KNN implementation against the modified KNN version. We consider that this work contributes to the implementation of machine learning techniques in real applications.

The document is organized as follows: Section 2 presents an architecture that implements the scalable KNN algorithm to process multiple inputs in parallel in independent cores and presents a parameterized design with the possibility of use for multiple modulation formats. We propose modifying the KNN algorithm to reduce operations in m-QAM demodulators with a square shape in Section 3. Section 4 shows the experimental setup for data acquisition used to validate the design, the setup for hardware implementation on a low-cost FPGA, and a high-performance FPGA. In Section 5, we show the results of the comparison between Matlab and FPGA implementation and the performance measures in terms of computation time and hardware resources. Finally, we present the conclusions of the work in Section 6.

2. Scalable Multicore KNN-Based Demodulator

The KNN algorithm is a machine learning technique that consists of the comparison of the input data against data previously labeled with a specific class, called training data. This comparison is made by measuring the distance between the input data and the trained data. The training data with the lowest distance values are known as the nearest neighbors. The label that repeats the most among the K nearest neighbors is assigned to the input data. Algorithm 1 shows the execution of the KNN algorithm in a sequential computation. The algorithm starts with the loading of the training data with a length of L. The input of the KNN algorithm are symbols that come from a channel with m-QAM modulation, so each symbol is represented by a pair (I, Q). Each data point contains three values: In-phase (I), quadrature (Q), and the *label*, which identifies that data point in a cluster of the m-QAM constellation. Also, the number of nearest neighbors used in the algorithm is chosen and stored in the variable K.

The Euclidean distance between the input symbol and all training data is calculated using the following expression: $d(i) = \sqrt{(I - I_t(i))^2 + (Q - Q_t(i))^2}$, where the subindex t indicates that it belongs to the training data, and i is the array index taking values in the range 1 to L. Please note that the distance calculation must be done L times. In a hardware implementation, the square root is not calculated to reduce operations. The distances should be sorted in an ascending order to select the smallest distances. The *labels* should be sorted simultaneously to get the K labels for the nearest neighbors. The execution time of the sorting algorithms depends on the distribution of data, but time complexity is well known. In this work, we use the Merge Sort algorithm, whose time complexity is $\mathcal{O}(n \log n)$. The distance array and the label array to be sorted are of length L.

Algorithm 1 describes the pseudocode for the sequential implementation of KNN-based m-QAM demodulators. However, a hardware-based approach is necessary for accelerating the algorithm in a parallel architecture, enhancing the particularities of the m-QAM modulations and the independence in the operations of the KNN algorithm. For this, we define custom processing units.

Algorithm 1: KNN algorithm for demodulators

Initialization;
Load L training data (I, Q, *label*);
Choose the value of K;
Input: Symbols (I, Q);
Output: Labels of the symbols;
for *each input symbol* (I, Q) **do**
  step 1. Find the Euclidean distance to all training data L points;
  step 2. Sort the distances and the training data labels in ascending order;
  step 3. Selects the first K values of the sorted labels. Label the input symbol with the modal value;
end

2.1. Custom Processing Units for KNN-Based Demodulator

The custom processing units (CuPUs) helps to define more complex functions or high-level DSP operations. In hardware design the term Processing Element (PE) is often used as those elements that perform clock-independent operations and simple memoryless mapping in signal processing such as: add/sub, add/sub-and-shift, multiply, multiply-and-accumulate, butterfly, etc. [26]. The CuPUs are not simple arithmetic operations on input data. They are complete routines that would imply in RTL an FSM with datapath. In this sense, the modules are closer to processing units than to processing elements. The word Custom was added to specify that these units already have optimization directives previously defined by the authors. The CuPUs perform basic independent operations, allowing the use of multiple components in parallel on hardware to improve the performance of the algorithm. For the KNN algorithm, a Calculate Distances CuPu calculates L distances between the symbols and the training data. The L training labels are sorted in ascending order according to the distance by a Sort Distances CuPU. Finally, the first K labels are selected from the sorted label array, and the input symbol is labeled with the modal value, also inside a Calculate Mode CuPU.

2.2. Scalable Multicore Architecture Based on Cupus

We propose a scalable multicore architecture based on CuPUs to improve the performance of the KNN algorithm. This architecture was designed for field-programmable gate array (FPGA), seeking to exploit the inherent parallelism that characterizes them. Scalable design is intended to be used in different FPGA targets. We define a parameter N as the number of cores of each of the CuPUs used in the design, i.e., $N = 3$ implies using 3 Calculate Distances CuPU, 3 Sort Distance CuPU, and 3 Calculate Mode CuPU, see Figure 1.

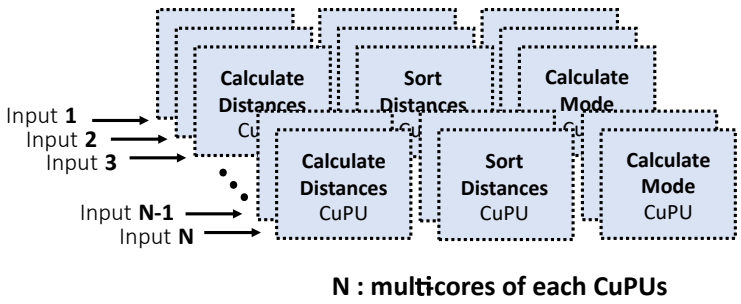

Figure 1. A general architecture for the KNN algorithm using N numbers of CuPUs in parallel.

We define N as the number of symbols entering the CuPUs for processing in parallel. Optimization directives such as pipelining, unrolling, and partitioning are added to the

CuPUs to increase operational concurrency and improve performance. All High-Level Synthesis (HLS) tools use optimization directives for algorithm acceleration on hardware.

In a communications system, a transmitter sends symbols through a single channel or multiple channels to the receiver. For this reason, our design stores the symbol streams in buffers for separating into N independent symbols for parallel processing in different CuPUs, see Figure 2. The independence in the operations on each input symbol in the KNN algorithm allows the parallel processing of multiple channels using the same training data. In turn, each channel can be processed on multiple cores, expressing this as $N = C \times N_C$, where C is the number of channels, and N_C is the number of cores per channel.

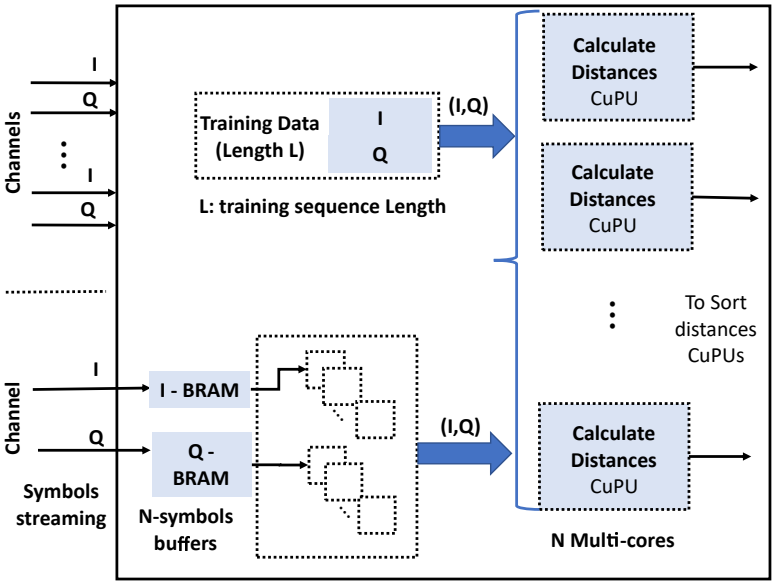

Figure 2. Calculate distance CuPUs by processing N symbols in parallel. The buffer converts the serial data in parallel.

A Calculate Distance CuPU computes the distance between each symbol and the L training samples. The L distance calculations applied to each symbol are also independent operations allowing unrolling operation on the loop. The unroll directive increases the consumption of hardware resources, and when resources are limited, it is convenient to perform a partial unroll. The combination of unrolling and the pipeline directive allows the reuse of resources.

After the distance calculation, it is necessary to sort the L values of the distance array in the Sort Distance CuPU. The sorting algorithm used is a Restructured Merge Sort [27], and it is a sorting algorithm optimized through directives in HLS for implementation in FPGA. The training data labels are sorted simultaneously with the distance array. N label array copies guarantee the parallel processing in the N cores, see Figure 3.

The Calculation Mode CuPU selects the first K labels from the sorted array, see Figure 3. The modal value serves to assign the right cluster of the constellation to the input symbol. In m-QAM schemes, there are m different labels.

This section presented the design of a scalable multicore architecture based on CuPUs. The architecture considers the particularities of communication systems when implementing real-time demodulators based on the KNN algorithm on FPGAs. The design has a focus on m-QAM, but it is not limited to this type of modulation. In the following section, we propose a modified version of the KNN algorithm to reduce the number of operations bringing the focus closer to m-QAM based on the constellation shape.

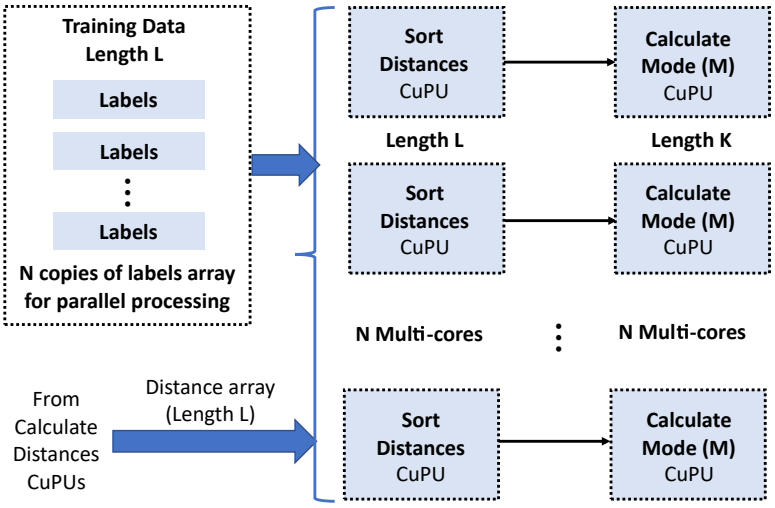

Figure 3. Sort Distances and Calculate Mode CuPUs by processing N symbols. The labels are copied N times to be operated in parallel.

3. Modified KNN Algorithm to Reduce Operations in M-Qam

The implementation of the KNN algorithm presented in the previous section can be modified to take advantage of the symbol geometric distribution seen in a constellation diagram. QAM schemes have characteristics that differentiate them from other applications that use the KNN algorithm. In this case, the inputs are symbols represented as a pair of real numbers. The symbols arrive at the receiver in a different position inside constellation. It happens because of noise and distortions in the travel through the channel. The recovery possibility of the symbols depends on how far they are from the initial position. In a successful transmission, we expect the symbols in certain zones of the constellation to consider recoverable. We propose the following hypothesis:

Hypothesis (H1): *For an input symbol to be correctly classified using the KNN algorithm, it must belong to the cluster-block to which the nearest ideal cluster belongs.*

Moreover, we present the following definition:

***Cluster-Block**: It is an 8-connected neighborhood of nine clusters: a central cluster and the eight clusters surrounding it.*

Algorithm 2 shows the modified KNN for m-QAM. The algorithm adds two steps before step 1, seen in Algorithm 1. Step 0.1 calculates the Euclidean distance between the input symbol and the m clusters of the ideal constellation. According to our hypothesis, the cluster with the closest distance or one of the 8-connected neighbors contains the input symbol label. Because of this, in step 0.2, we carry out the labels corresponding to the cluster-block. In this modified KNN algorithm, the training data are organized by labels and and the memory addressing is established to access only the positions corresponding to the selected Cluster-Block.

The algorithm considerably reduces operations by using only the training data associated with a cluster-block. Figure 4 shows a square-shaped constellation of size $m \geq 8$. The cluster-block size varies depending on the constellation position, but we always have a size of 4 clusters for the blocks at the corners, 6 clusters for the blocks at the edges, and 9 clusters for the blocks in the center of the constellation. The figure shows an example of the application of cluster-blocks on a constellation diagram for 16-QAM. It also follows that for constellations without corners, we would have values of 5 and 8 clusters for the blocks near the corners. The other sizes are 6 and 9 clusters, as in the constellations with corners.

Algorithm 2: Modified KNN algorithm for m-QAM demodulators

Initialization;
Load L training data (I, Q, *label*) grouped by label;
Choose the value of K;
Input: Symbols (I, Q);
Output: Labels of the symbols;
for *each input symbol* (I, Q) **do**
    step 0.1. Calculate the distances between the input symbol and the m clusters of the ideal constellation;
    step 0.2. Select the training data belonging to the nearest ideal cluster and its cluster-block;
    step 1, step 2, step 3 as seen in Algorithm 1;
end

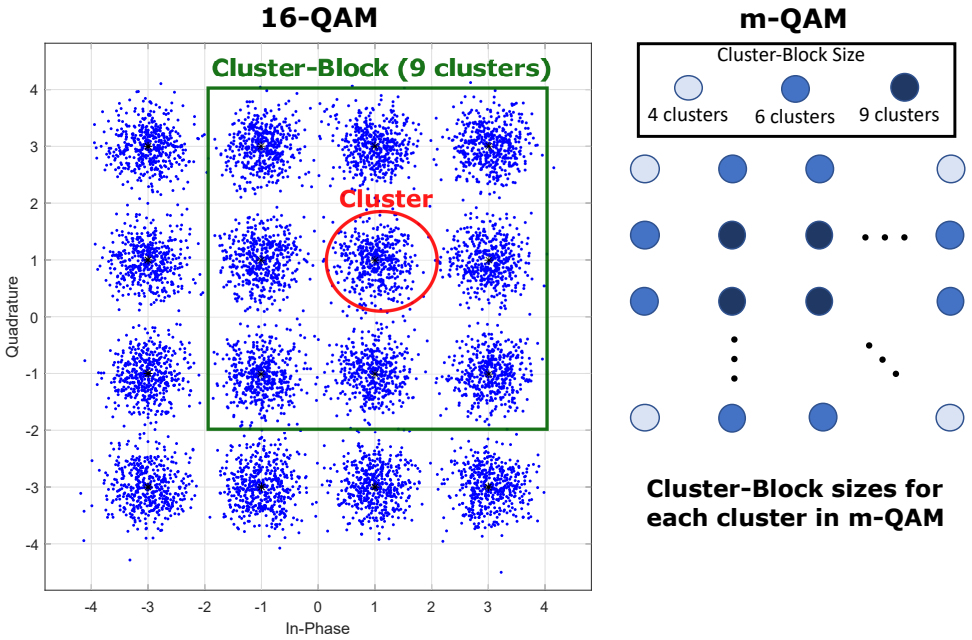

Figure 4. On the left side, a cluster-block example in a 16-QAM for the modified KNN algorithm, according to the 8-conected neighbors rule. On the right side, the generalization of each cluster-block's sizes in an m-QAM constellation, the only possible cluster-block sizes are 4, 6, or 9.

The algorithm reduces the L training data in Calculate Distance CuPU and Sort Distance CuPU. In this case, we express the total length of training data as $L = Lc \times m$, where L_c is the length of the training data for a cluster, and remember that $m \geq 8$. In this way, L is much bigger than the number of training data for the biggest cluster-block.

Calculate Mode CuPU depends on the value of K, and the reduction in L does not affect it, but this CuPU is the least amount of operations performed within the KNN algorithm because it assumes $K << L$.

Algorithm 2 implementation on FPGA uses the CuPUs as in Algorithm 2, but with a smaller training data length due to reduction of operations, as described in this section, i.e., we propose the implementation of the modified KNN algorithm using a scalable multicore architecture to process the input symbols in parallel, as in Algorithm 1. The CuPUs were implemented with optimization directives to increase the concurrency of operations within each processing unit.

Figure 5 shows the architecture of the algorithm for the modified KNN. A new processing unit is attached to performs the Cluster-Block assignment to the input symbols. This block measures the distance to the ideal constellation and selects the training data taking only those belonging to the Cluster-Block.

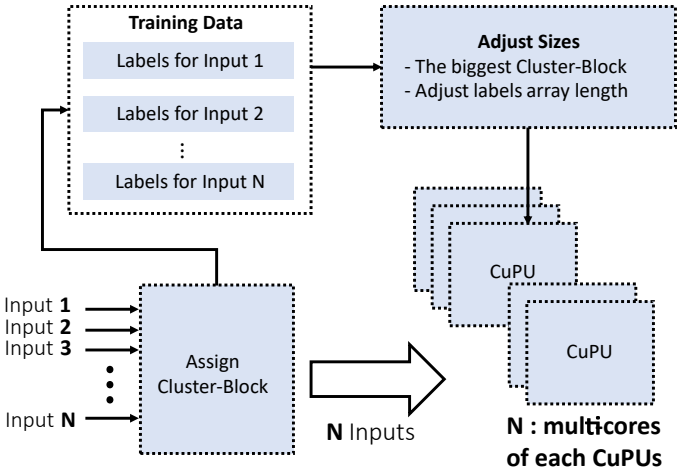

Figure 5. Hardware architecture for the implementation of the modified KNN algorithm. New blocks were added to the original KNN architecture.

The CuPUs are built to independently compute N inputs of the KNN algorithm taking full advantage of the parallelism. For the same function, the architecture uses copies of the same CuPU that are executed in parallel. An implementation of Algorithm 2 using the previously designed multi-core architecture requires adding a module to adjust the training data sizes for each symbol. In this version, each of the N symbols is assigned to a Cluster-Block with three different size options (4, 6, 9). Additionally, the amount of training data for each tag may vary depending on the application. A version without size adjustment would eliminate the synchronization of the CuPUs with outputs at different times, depending on the input symbol's value. In this work, we want to take advantage of the highly parallelized and parameterized architecture of Algorithm 1, so we adjusted the data size to the largest value for the N label arrays. However, we hope in future versions to design a structure that includes the synchronization stage to take advantage of the reduction of operations that allows the modified KNN algorithm.

4. Experimental Setup

4.1. Validation in Gridless Nyquist-WDM System Affected by ICI

We validate the KNN-based demodulator in a Gridless Nyquist-WDM system affected by Inter-Channel Interference (ICI) [15]. The authors used this experimental setup to collect real data in gridless scenarios with ICI. Then, the authors process the data offline in MATLAB to find the optimal KNN parameters. The authors plotted K-neighbors against the bit error rate (BER), finding a value of 15 for K. Also, using BER, they found a value of 3240 for the training data set.

In the experimental setup, three lasers with 100 kHz linewidths generated 3×16 Gbaud 16-QAM Nyquist WDM system. Different DACs operated at 64 Gsamp/s generated the center and side channels to produce uncorrelated signals, see Figure 6. The authors used a random bit sequence and mapped it to generate 16-QAM symbols using a root raised cosine filter with a roll-off factor of 0.1. They added optical noise to yield an OSNR from 19.3 dB to 31.3 dB. The receiver used a wideband coherent receiver with an electrical bandwidth of 35 GHz. The signal was digitized by an 80 Gsamp/s real-time oscilloscope, followed by offline post-detection DSP processing in MATLAB.

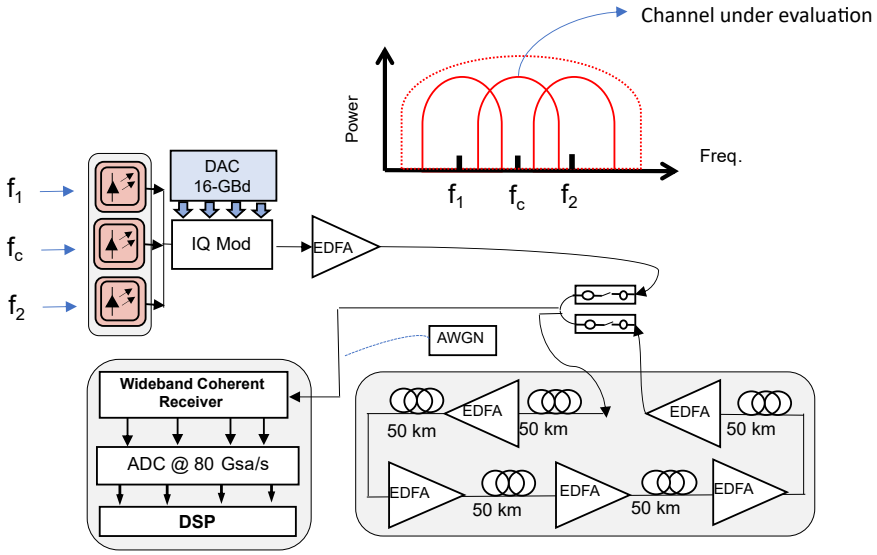

Figure 6. Experimental setup of the Gridless Nyquist-WDM system affected by Inter-Channel Interference (ICI).

4.2. Real-Time Implementation of KNN-Based Demodulator on Low-Cost FPGA

The implementation of the algorithms was done on a low-cost and high-performance platform. The design space exploration was performed on a low-cost architecture because this stage implies long development times. Also, the designs are small scale with relatively short synthesis times, allowing a broad exploration and verification of the algorithm's parameterization. Additionally, the high-performance platform used in this work was AWS-FPGA, which implies costs by use.

The selected low-cost platform was the Ultra96. The board has a chip Xilinx Zynq UltraScale+ MPSoC ZU3EG A484 with a heterogeneous architecture integrating a Quad-core ARM Cortex-A53 MPCore microprocesso and programmable logic based on FPGA. Specifically, 141 K Flip-Flops (FF), 71 K Lookup Table (LUT), 1.8 M of maximum distributed RAM, and 360 DSP slices. This architecture facilitates the comparison with the offline experimental results because the ARM quickly loads the stored data. The microprocessor read the training data and the symbols from the files and controls the start and the stop of the KNN IP core. The KNN was implemented in the FPGA using Vivado HLS (High-Level Synthesis).

4.3. Real-Time Implementation of KNN-Based Demodulator on Aws-FPGA

The migration of the design to a high-performance architecture allows scaling the KNN algorithm implementations. Amazon EC2 F1 provides a platform for design implementation on a Xilinx Virtex UltraScale+ VU9P FPGAs. The device, manufactured with 16 nm technology, has approximately 2.5 million logic cells, more than 6800 Digital Signal Processing (DSP) engines. Additionally, four DDR4 channels providing a total bandwidth of 48 Gb per second. The high speed of the data communication allows emulating the communication channels using the PCI-e as the ADC in a communication system. The sampling frequency of the symbols depends on the numerical format and the maximum reading speed that can be achieved by the implementation in the hardware.

5. Results and Discussion

5.1. Validation Using the Experimental Setup

Figure 7 shows the bit error rate versus the optical signal to noise ratio (OSNR) for both the offline MATLAB obtained in [15] and real-time FPGA implementation of KNN. The dotted lines are the FPGA results, and the continuous lines are Matlab results. The colors of the lines represent different channel spacing. Matlab implementation used a 64-bits double-precision format for the data. Instead, the FPGA implementation used a

32-bit single-precision floating-point format; nonetheless, the difference is negligible below FEC limit.

Table 1 shows the characteristics of the experimental setup data. The sizes of the training data set for each label are different and total of 3240. The column Cluster-Blocks presents the clusters for each of the labels. In this way, it is possible to determine the number of operations that should be carried out with the training data, such as calculating distances and orders. The reduction in operations is significant for all sizes of Cluster-Block. The reduction percentage for 4-size clusters was 75%; for 6-size, it was 62.4% and 43.4% for 9-size Cluster-Block.

Table 1. Reduction of operations using the modified KNN algorithm. The table shows the application of the cluster-blocks on the real data of the experimental setup.

16-QAM Labels	Training Data	Cluster-Block	Number of Operation	% Reduction
0	204	0, 1, 4, 5	798	75.37
1	195	1, 0, 2, 4, 5, 6	1209	62.69
2	210	2, 1, 3, 5, 6, 7	1227	62.13
3	195	3, 2, 6, 7	811	74.97
4	178	4, 0, 1, 5, 8, 9	1231	62.01
5	221	5, 0, 1, 2, 4, 6, 8, 9, 10	1834	43.40
6	201	6, 1, 2, 3, 5, 7, 9, 10, 11	1838	43.27
7	205	7, 2, 3, 6, 10, 11	1211	62.62
8	222	8, 4, 5, 9, 12, 13	1220	62.35
9	211	9, 4, 5, 6, 8, 10, 12, 13, 14	1812	44.07
10	192	10, 5, 6, 7, 9, 11, 13, 14, 15	1849	42.93
11	208	11, 6, 7, 10, 14, 15	1216	62.47
12	187	12, 8, 9, 13	821	74.66
13	201	13, 8, 9, 10, 12, 14	1212	62.59
14	199	14, 9, 10, 11, 13, 15	1222	62.28
15	211	15, 10, 11, 14	810	75.00
Total	3240			

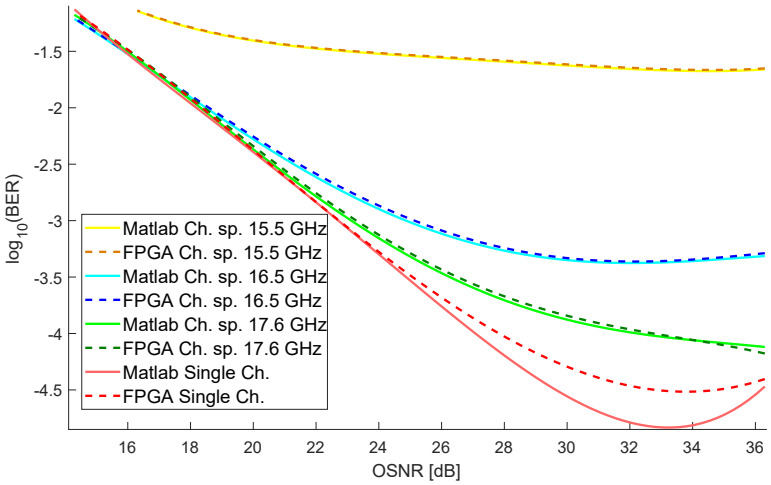

Figure 7. Comparison of results between Matlab using 64-bits and the FPGA using 32-bits.

5.2. Multi-Core Performance Testing and Scaling on Low-Cost FPGA

The KNN parameters in the experiment were 15 nearest neighbors, 3240 samples of training data, and 100,000 symbols modulated in 16-QAM. The Ultra96 performed the KNN-based demodulator with 1 to 8 cores. Figure 8 shows the total processing time on

the Ultra96 performing the implementations with a clock of 187.5 MHz. We achieved a speed-up of six times when using 8 instead of 1 core.

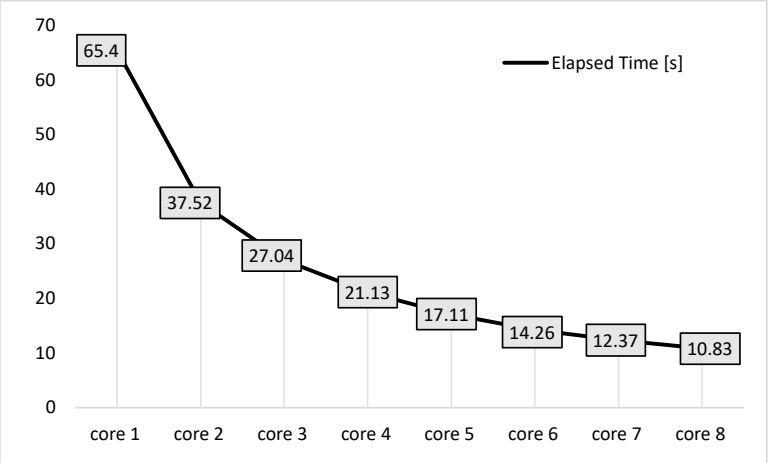

Figure 8. Total computation time for 100k input symbols using a different number of cores.

Table 2 shows the hardware resources consumed. We can see the LookUp Tables LUTs, Flip flops, Block RAM, and DSPs in the table. In this experiment, the most used hardware resource was Block RAM, see Figure 9. In the design space exploration, we applied the optimization directives to reach the highest number of cores in the low-cost Ultra96.

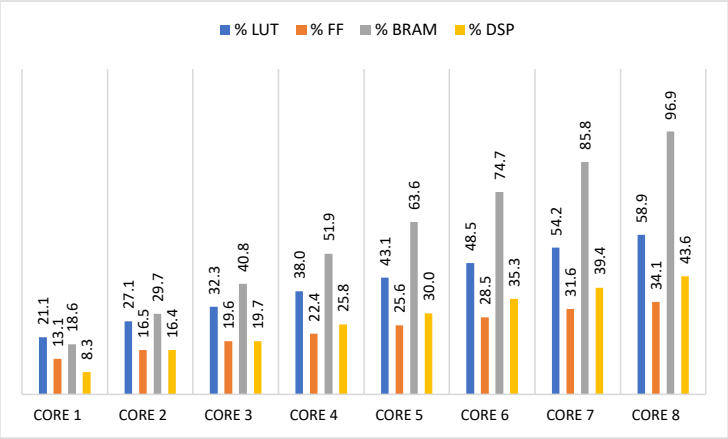

Figure 9. The graph shows the percentage of resources consumed when increasing the number of cores on the Ultra96 board.

Table 2. Hardware resource consumption using a different number of cores on Ultra96 board.

N-Cores	BRAM	FF	LUT	DSP
1	33.5	18,465	14,985	30
2	53.5	23,238	19,229	59
3	73.5	27,640	22,959	71
4	93.5	31,586	27,004	93
5	114.5	36,084	30,614	108
6	134.5	40,250	34,422	127
7	154.5	44,592	38,454	142
8	174.5	48,116	41,794	157

5.3. Implementation on the High-Performance AWS-FPGA

The AWS-FPGA platform significantly outperforms Ultra96 in technology and hardware resources. Xilinx Virtex UltraScale+ VU9P on the high performance platform has 16 times more logic cells and 18 times more DSPs than Ultra96. However, the exploration of the design space allowed scaling the architecture to obtain a higher number of cores on the AWS-FPGA. In this way, we perform the algorithm's scaling up to 20 cores of the KNN and the modified KNN. The results in terms of hardware resource consumption and processing time are shown in Table 3.

Table 3. Comparison between the KNN algorithm and the modified KNN algorithm for 100 K input symbols using 20 cores on the AWS-FPGA.

	N-Cores	BRAM	FF	LUT	DSP	Elapsed Time (s)
KNN	20	369.5	242,996	177,445	75	5.82
Modified KNN	20	403.5	107,108	99,759	321	3.07

The elapsed time is for processing 100 K symbols i.e., 17,182 symbols per second on our multi-core KNN architecture. The architecture for our modified KNN algorithm processes 32,573 symbols per second, improving performance by 1.9 times. The implementations used a 32-bit floating-point format so that the results were compared with those obtained by the authors in Matlab, achieving 1.04 Mb/s for the modified KNN. Nevertheless, the number of bits per symbol from an ADC for optical communications applications is usually much lower, so internal operations in CuPUs could be performed using a fixed-point format with a lower number of bits. The percentage of hardware resources over the AWS-FPGA is low for the 20-core design, so it is possible to explore a larger design that processes more input symbols in parallel.

6. Conclusions

In this work, we presented a multi-core architecture for real-time implementation of KNN-based demodulators. The design takes advantage of the operations' independence applied to the symbols coming from a communication channel. The design allowed selecting N input symbols to configure N parallel processing cores. These cores contain custom processing units (CuPUs) implemented with optimization directives to increase concurrency within each core. The implementation was performed using a high-level synthesis (HLS) design methodology, simplifying the selection of m-QAM formats and the KNN parameters. This version of KNN could be used in formats other than m-QAM modulation because KNN results depend on the training data and not on the constellation's shape.

Additionally, we proposed a modification of the KNN algorithm to improve the performance of the hardware implementation. In this case, the modulation format does concentrate on square-shaped m-QAM constellations. We define Cluster-Blocks to significantly reduce the number of comparisons and sortings on the training data. The modified KNN algorithm shows a decrease in computation times concerning the first KNN version. For the hardware implementation, we use as a base the multi-core architecture to preserve the advantages in scaling and parameterization.

For future work, we propose testing KNN implementations with data from other experimental setups and different modulation formats. In addition, we suggest increasing the design exploration for the KNN algorithms to obtain better processing times, for example, exploring the impact on the use of fixed-point numerical formats with a lower amount of bits or efficient memory management techniques. Finally, we propose exploring this type of modification in other machine learning algorithms.

Author Contributions: Conceptualization, D.M.-V. and N.G.-G.; methodology D.M.-V., L.C.-L., and N.G.-G.; validation, D.M.-V. and L.C.-L.; formal analysis, D.M.-V. and N.G.-G.; investigation, D.M.-V., L.C.-L., and N.G.-G.; writing—original draft preparation, D.M.-V.; writing-review and editing, D.M.-V., L.C.-L., and N.G.-G.; funding acquisition, D.M.-V. and L.C.-L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Instituto Tecnológico Metropolitano grant number P17224.

Data Availability Statement: Not applicable.

Acknowledgments: This work was developed in the LSCR laboratory of the Instituto Tecnológico Metropolitano.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chagnon, M. Optical Communications for Short Reach. *J. Light. Technol.* **2019**, *37*, 1779–1797. [CrossRef]
2. Wang, X.; Zhang, Q.; Xin, X.; Gao, R.; Tian, Q.; Tian, F.; Wang, C.; Pan, X.; Wang, Y.; Yang, L. Robust weighted K-means clustering algorithm for a probabilistic-shaped 64QAM coherent optical communication system. *Opt. Express* **2019**, *27*, 37601–37613. [CrossRef] [PubMed]
3. Chien, H.; Yu, J.; Cai, Y.; Zhu, B.; Xiao, X.; Xia, Y.; Wei, X.; Wang, T.; Chen, Y. Approaching Terabits Per Carrier Metro-Regional Transmission Using Beyond-100GBd Coherent Optics With Probabilistically Shaped DP-64QAM Modulation. *J. Light. Technol.* **2019**, *37*, 1751–1755. [CrossRef]
4. Wang, D.; Zhang, M.; Li, J.; Li, Z.; Li, J.; Song, C.; Chen, X. Intelligent constellation diagram analyzer using convolutional neural network-based deep learning. *Opt. Express* **2017**, *25*, 17150–17166. [CrossRef] [PubMed]
5. Fernández, E.A.; Torres, J.J.G.; Soto, A.M.C.; Gonzalez, N.G. Radio-over-Fiber signal demodulation in the presence of non-Gaussian distortions based on subregion constellation processing. *Opt. Fiber Technol.* **2019**, *53*, 102062. [CrossRef]
6. Lau, A.P.T.; Kahn, J.M. Signal Design and Detection in Presence of Nonlinear Phase Noise. *J. Light. Technol.* **2007**, *25*, 3008–3016. [CrossRef]
7. Han, Y.; Yu, S.; Li, M.; Yang, J.; Gu, W. An SVM-Based Detection for Coherent Optical APSK Systems With Nonlinear Phase Noise. *IEEE Photon. J.* **2014**, *6*, 1–10. [CrossRef]
8. Wang, D.; Zhang, M.; Fu, M.; Cai, Z.; Li, Z.; Han, H.; Cui, Y.; Luo, B. Nonlinearity Mitigation Using a Machine Learning Detector Based on *k* -Nearest Neighbors. *IEEE Photon. Technol. Lett.* **2016**, *28*, 2102–2105. [CrossRef]
9. Zhang, J.; Chen, W.; Gao, M.; Ma, Y.; Zhao, Y.; Chen, W.; Shen, G. Intelligent adaptive coherent optical receiver based on convolutional neural network and clustering algorithm. *Opt. Express* **2018**, *26*, 18684–18698. [CrossRef] [PubMed]
10. Wang, Y.; Liu, M.; Yang, J.; Gui, G. Data-Driven Deep Learning for Automatic Modulation Recognition in Cognitive Radios. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4074–4077. [CrossRef]
11. Jiang, K.; Zhang, J.; Wu, H.; Wang, A.; Iwahori, Y. A Novel Digital Modulation Recognition Algorithm Based on Deep Convolutional Neural Network. *Appl. Sci.* **2020**, *10*. [CrossRef]
12. Zhang, S.; Yaman, F.; Nakamura, K.; Inoue, T.; Kamalov, V.; Jovanovski, L.; Vusirikala, V.; Mateo, E.; Inada, Y.; Wang, T. Field and lab experimental demonstration of nonlinear impairment compensation using neural networks. *Nat. Commun.* **2019**, *10*, 1–8. [CrossRef] [PubMed]
13. Sun, L.; Du, J.; He, Z. Machine Learning for Nonlinearity Mitigation in CAP Modulated Optical Interconnect System by Using K-Nearest Neighbour Algorithm. In Proceedings of the 2016 Asia Communications and Photonics Conference (ACP), Wuhan, China, 2–5 November 2016; pp. 1–3.
14. Rottondi, C.; Barletta, L.; Giusti, A.; Tornatore, M. Machine-Learning Method for Quality of Transmission Prediction of Unestablished Lightpaths. *J. Opt. Commun. Netw.* **2018**, *10*, A286–A297. [CrossRef]
15. Pérez, A.E.; Torres, J.J.G.; González, N.G. KNN-based Demodulation in gridless Nyquist-WDM Systems affected by Interchannel Interference. In Proceedings of the OSA Advanced Photonics Congress (AP) 2019 (IPR, Networks, NOMA, SPPCom, PVLED), Burlingame, CA, USA, 29 July–1 August 2019; p. SpTh1E.3.
16. Asano, S.; Maruyama, T.; Yamaguchi, Y. Performance comparison of FPGA, GPU and CPU in image processing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 126–131. [CrossRef]
17. Gankidi, P.R.; Thangavelautham, J. FPGA architecture for deep learning and its application to planetary robotics. In Proceedings of the 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 4–11 March 2017; pp. 1–9. [CrossRef]
18. Zhang, M.; Li, L.; Wang, H.; Liu, Y.; Qin, H.; Zhao, W. Optimized Compression for Implementing Convolutional Neural Networks on FPGA. *Electronics* **2019**, *8*. [CrossRef]
19. Kashino, R.; Kobayashi, R.; Fujita, N.; Boku, T. Performance Evaluation of OpenCL-Enabled Inter-FPGA Optical Link Communication Framework CIRCUS and SMI. In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, Fukuoka, Japan, 15–17 January 2020; Association for Computing Machinery: New York, NY, USA, 2021; pp. 23–31. [CrossRef]

20. Pu, Y.; Peng, J.; Huang, L.; Chen, J. An Efficient KNN Algorithm Implemented on FPGA Based Heterogeneous Computing System Using OpenCL. In Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, 2–6 May 2015; pp. 167–170. [CrossRef]
21. Song, X.; Xie, T.; Fischer, S. A Memory-Access-Efficient Adaptive Implementation of kNN on FPGA through HLS. In Proceedings of the 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 17–20 November 2019; pp. 177–180. [CrossRef]
22. Sohrabizadeh, A.; Yu, C.H.; Gao, M.; Cong, J. AutoDSE: Enabling Software Programmers Design Efficient FPGA Accelerators. *arXiv* **2020**, arXiv:cs.AR/2009.14381.
23. de Fine Licht, J.; Besta, M.; Meierhans, S.; Hoefler, T. Transformations of High-Level Synthesis Codes for High-Performance Computing. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1014–1029. [CrossRef]
24. Vieira, J.; Duarte, R.P.; Neto, H.C. kNN-STUFF: kNN STreaming Unit for Fpgas. *IEEE Access* **2019**, *7*, 170864–170877. [CrossRef]
25. Stamoulias, I.; Manolakos, E.S. Parallel Architectures for the KNN Classifier–Design of Soft IP Cores and FPGA Implementations. *ACM Trans. Embed. Comput. Syst.* **2013**, *13*. [CrossRef]
26. Wanhammar, L. *DSP Integrated Circuits*; Academic Press Series in Engineering; Academic Press: Burlington, MA, USA, 1999; p. xiii. [CrossRef]
27. Kastner, R.; Matai, J.; Neuendorffer, S. Parallel Programming for FPGAs. *arXiv* **2018**, arXiv:cs.AR/1805.03648.