Enhancing Performance of Gabriel Graph-Based Classifiers by a Hardware Co-Processor for Embedded System Applications

Janier Arias-Garcia, Augusto Mafra, Liliane Gade, Frederico Coelho, Cristiano Castro, Luiz Torres, and Antônio Braga

***Abstract*—It is well known that there is an increasing interest in edge computing to reduce the distance between cloud and end devices, especially for machine learning (ML) methods. However, when related to latency-sensitive applications, little work can be found in ML literature on suitable embedded systems implementations. This article presents new ways to implement the decision rule of a large margin classifier based on Gabriel graphs as well as an efficient implementation of this on an embedded system. The proposed approach uses the nearest neighbor method as the decision rule, and the implementation starts from an RTL pipeline architecture developed for binary large margin classifiers and proposes the integration in a hardware/software co-design. Results showed that the proposed approach was statistically similar to the classifier and had a speedup factor of up to eight times compared to the classifier executed in software, with performance suitable for ML latency-sensitive applications.**

***Index Terms*—Classifiers, embedded system, FPGA, Gabriel graph, IoT, large margin, latency-sensitive applications, machine learning, system-on-a-chip.**

I. Introduction

CURRENTLY, the interest in edge computing is increasing in order to reduce the distance between the cloud and the end devices, especially when related to latency-sensitive applications [1]. In parallel, research on Machine Learning (ML) methods is boosted by the need for automated decisions at the presence of the massive amount of data available in the industry (Industry 4.0). However, the main challenges are related to the capabilities of the electronic devices to deal with the real industrial system (IS) implementing ML methods to comply with electronic design parameters as well as IS requirements such as security, robustness, accessibility, real time processing, and battery life among others [2]–[5].

Recent literature in the area of embedded applications highlights the tendency in IoT applications to perform the processing on the devices connected to the network in the paradigm called Edge Computing. Shi and Dustdar [6] emphasize the promising perspective of this computing model as a reaction to the Cloud Computing paradigm, in which processing takes place entirely on servers in the network. In this new scenario, devices at the network edge also perform processing, storage, caching, and load balancing operations. Furthermore, CISCO coined Fog Computing as a novel parading that enables provisioning resources and services outside the cloud, at the edge of the network. This means that the cloud would be closer to end devices, which should be suitable to most IoT applications, involving latency-sensitive applications [1]. Also, the application of embedded technologies becomes relevant with a higher production capacity of high-end IoT systems, and Big-Data in the industrial environment and in consumer goods [3], [4].

From these two trends in the current technological cycle, the use of ML in embedded platforms appears as a promising approach (for instance, for predictive maintenance [7]). While IoT applications depend on the development and interconnection of embedded computing systems, manipulation and inference of large-volume of information in BigData remains associated with large-server computing because of the complexity and cost of algorithms and data involved. For this reason, applications of this category directly in hardware compose a branch of research still little explored and with ample possibilities of technological development.

In this context, embedded technologies, and in particular FPGA accelerator platforms, have received increasing attention in the academic and industrial milieu since the 1990s [8]. The major attractions for the implementation in this type of platform are the performance and parallelism made possible by the hardware. At the same time, its reprogrammable characteristic confers greater flexibility to the applications.

Manuscript received January 16, 2020; accepted March 31, 2020. Date of publication April 20, 2020; date of current version November 18, 2020.

Janier Arias-Garcia and Antônio Braga are with the Graduate Program in Electrical Engineering, Department of Electronic Engineering, *Universidade Federal de Minas Gerais* (UFMG), Belo Horizonte, MG 31270-901, Brazil (e-mail: janier-arias@ufmg.br).

Augusto Mafra, Liliane Gade, and Frederico Coelho are with the Department of Electronic Engineering, Universidade Federal de Minas Gerais, Belo Horizonte 31270-901, Brazil (e-mail: augustomafra1@gmail.com; liligade4@gmail.com; fredgfc@ufmg.br).

Cristiano Castro is with the Graduate Program in Electrical Engineering and with the Department of Electric Engineering, *Universidade Federal de Minas Gerais* (UFMG), Belo Horizonte 31270-901 Brazil (e-mail: crislcastro@ufmg.br).

Luiz Torres is with the Department of Computing and Systems, Universidade Federal de Ouro Preto, João Monlevade 35931-022, Brazil (e-mail: luiz.torres@ufop.edu.br).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2020.2987329

1551-3203 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

Specifically, in embedded ML applications, studies describe efficient implementations of accelerators that rely on parallelism architectures and cascading connections. Papadonikolakis and Bouganis[9] detail an FPGA architecture for a support vector machines (SVM) classifier that uses dedicated arithmetic units to obtain an acceleration factor up to 7 times higher than other implementations in software or accelerated with Graphical Processing Units (GPUs). In another approach, Gong *et al.* [10] describes an architecture optimized for high performance for a Convolutional Neural Network (CNN) method in FPGA. Its application is designed over structures with pipeline, parallelism and data transfer modules to achieve high throughput in a dedicated processor. In all models proposed for the acceleration of ML algorithms, the communication interfaces between the implemented modules are highlighted in the architecture and incorrect sizing can lead to bottlenecks in the system that prevent the accelerator from achieving the desired performance.

In that order of ideas, there is also some applications using SoC, like in [11], which proposes a 65nm SoC to act as a near-sensor data analytics for IoT end-nodes. This system was developed using architecture solution combining accelerators and cores in a cluster. Using the same principle, Pullini *et al.* [12] also proposed a SoC that integrates a Convolutional Neural Network with a near-threshold parallel processor cluster. Ortega-Zamoran *et al.* [13] implemented the backpropagation learning algorithm in a FPGA. They introduced a new neuron representation reducing the resource usage by combining the input and first hidden layer units in a single module. In order to try to take advantage of the built-in digital signal processor cores, they implemented a time-division multiplexing scheme to carry out product computations. Data type representation was based on a fixed-point scheme in order to reduce system memory variable usage and to increase computation speed.

Hussain *et al.* [14] propose different adaptive FPGA architectures to implement the k-nearest neighbor algorithm (KNN), using different levels of parallelism. Three basic blocks are implemented: distance computation, Nearest neighbor finder and query label finder. On the other hand, [15] presented an FPGA acceleration system for Neural Network Q-learning (NNQL), with support to online parameterization that allows algorithms to restructure the network dynamically. Furthermore, [16] presents an application in the field of computer vision where a SoC system is well used to implement Machine Learning algorithms, and states that hardware architectures must be optimized to the entire SoC and not only to some accelerators.

Despite advances in FPGA-based accelerator technology, this platform remains less utilized than its counterpart implemented in dedicated hardware such as GPUs amid scientific and commercial applications. This shortcoming is due to the lack of specific design methods and tools for FPGA/SoC applications, which still require advanced hardware design techniques [8].

Based on the previous discussion we can summarize the contributions of this article as follows.

1) A distance-based hardware classifier that does not require user parameter settings or brute-force methods such as grid-search and cross-validation, and offers results close to SVM (state of the art) without using quadratic programming methods.
2) Potential application of the proposed approach in Industry 4.0 to classify failures in real time due to high processing speed.
3) A new, more efficient architecture for intelligent systems that can be designed on cheaper hardware that consumes less power.
4) An analysis about the performance of the proposed embedded system.

The rest of this article is organized as follows. Section II introduces the learning algorithm for classification problems that is suitable for integrated circuit implementation. Section III presents the proposed embedded system architecture and in Section IV, the performance results are presented. Finally, Section V concludes this article.

II. Learning Algorithm

Torres *et al.* [17] presented a new learning method for classification problems, called *CHIP-clas*, that is suitable for integrated circuit implementation. The method is based on a structural description of the learning set represented by a planar graph known by *Gabriel graph* [18]. The final large margin classifier is composed of a mix of local experts. The method is also suitable for incremental and online learning since the parameters of the model are obtained directly from the data set based only on distance evaluation without the need for user interaction for the hyperparameter setting. Method implementation is accomplished using two basic steps: elimination of noisy data according to graph propertie, and detection of the separation boundaries between classes.

A. Distance-Based Large Margin Classifier

The Gabriel graph of a set of points consists of a graph whose set of vertices V and edges E must conform with the definition expressed by (1), where points $(\mathbf{p}_i, \mathbf{p}_j)$ constitute an edge if, and only if, no other point is contained within the hypersphere of diameter $D(\mathbf{p}_i, \mathbf{p}_j)$, the distance between points $\mathbf{p}_i$ and $\mathbf{p}_j$

$$(\mathbf{p}_i, \mathbf{p}_j) \in E \leftrightarrow D^2(\mathbf{p}_i, \mathbf{p}_j) \leq [D^2(\mathbf{p}_i, \mathbf{z}) + D^2(\mathbf{p}_j, \mathbf{z})]$$
$$\forall \mathbf{z} \in V, \mathbf{p}_i, \mathbf{p}_j \neq \mathbf{z}. \tag{1}$$

The graph $G = \{V, E\}$ of a training data set $T = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, N\}$, where $y_i \in \{+1, -1\}$ and $\mathbf{x}_i \in \mathbb{R}^n$ also has a set of support edges SE, which represents all the edges of E that have a pair of vertices $(\mathbf{x}_i, \mathbf{x}_j)$ that belong to opposite classes. If there is no overlap among patterns, it is possible to state that the vertices of SE are located in the separation margin between classes. Thus, the hyperplane H_l which passes through the midpoint of a pair of vertices $\mathbf{x}_i$ and $\mathbf{x}_j$ belonging to SE, refers to the maximal margin classifier in relation to the two support vertices [17].

If there is overlap between classes, a filtering technique based on low degree vertices [19] should eliminate noisy patterns. This filter is defined by (2), where $Gr(\mathbf{x}_i)$ represents the degree of vertex $\mathbf{x}_i$ and $\hat{Gr}(\mathbf{x}_i)$ the degree of $\mathbf{x}_i$ minus the number of

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

vertices of the opposite class

$$q(\mathbf{x}_i) = \frac{\hat{Gr}(\mathbf{x}_i)}{Gr(\mathbf{x}_i)}$$
$$\hat{Gr}(\mathbf{x}_i) = \{\forall \mathbf{x}_j \in Gr(\mathbf{x}_i) | y_j = y_i\}. \quad (2)$$

All vertices of G that have $q(\mathbf{x}_i)$ smaller than threshold t^+ and t^- for classes $+1$ and -1 respectively, are removed. In *CHIP-clas* filtering is only performed if $q(\mathbf{x}) > 0, \ \forall \mathbf{x} \in T$. t^+ and t^- are defined in (3) as the average of the quality measures Q^+ and Q^- within classes $+1$ and -1, obtained from $q(\mathbf{x}_i)$ values grouped by class

$$t^+ = \frac{\sum_{q(\mathbf{x}_i)\in Q^+} q(\mathbf{x}_i)}{|Q^+|}, \ t^- = \frac{\sum_{q(\mathbf{x}_i)\in Q^-} q(\mathbf{x}_i)}{|Q^-|}. \quad (3)$$

After filtering step, SE can be determined and a classifier will be generated represented by the hyperplane H_i that passes through the midpoint of such vertices.

B. Mixing Hyperplanes

The final classifier is obtained from a Hierarchical Expert Mixture (HME), in which each hyperplane associated with every SE will have a different weight according to the new input pattern $\mathbf{x}$ to be classified. According to [17], the HME architecture is represented by a network, where the first layer corresponds to the local experts. Each expert is weighted according to

$$c_l(\mathbf{x}) = \exp - \left(\frac{(\max(D(\mathbf{x}, p_k)))^2}{D(\mathbf{x}, p_l)}\right) \quad \forall k = 1, \ldots, m \quad (4)$$

functions $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$ represents the m specialist outputs, $pl = (\mathbf{x}_i + \mathbf{x}_j)/2$ is the midpoint of the support edge formed by the vertices $\mathbf{x}_i$ and $\mathbf{x}_j$ and $D(\mathbf{x}, p_l)$ is the distance between sample $\mathbf{x}$ (to be labeled) and the midpoint p_l.

After calculating the weights by (4) a normalization is imposed, such that $\sum_{l=1}^{m} c_l(\mathbf{x}) = 1$. Then, the final result of the classification is obtained by $f(\mathbf{x}) = \text{sign}(\sum_{l=1}^{m} h_l(\mathbf{x}) c_l(\mathbf{x}))$, where $h_l = \text{sign}(\mathbf{x}^T w_l - b_l)$, where sign(.) is the signal function, w_l is defined as $(\mathbf{x}_l - \mathbf{x}_j)$, and the bias term b_l is defined as $[(1/2)(\mathbf{x}_l + \mathbf{x}_j)] w_l^T$.

The mixture of hyperplanes is still costly for hardware implementation, mainly because of the exponential and $f(\mathbf{x})$ calculations in (4) thus a simplification is proposed next.

C. Reduced CHIP-Clas

A reduced version of the original CHIP-clas is obtained as follows: given a sample $\mathbf{x}$ to be labeled, find the midpoint l closest to $\mathbf{x}$ by means of the nearest hyperplane rule

$$\arg\min_l D(\mathbf{x}, P_l), \quad l = 1, \ldots, m \quad (5)$$

where $D(\cdot)$ is the distance between two vectors.

The separation hyperplane that passes through the midpoint l closest to $\mathbf{x}$ is defined as $H_l(\mathbf{x}) = (\mathbf{x}^T w_l - b_l)$, where $b_l = [(1/2)(\mathbf{x}_i + \mathbf{x}_j)] w_l^T$ and $w_l = (\mathbf{x}_i + \mathbf{x}_j)$, where $(\mathbf{x}_i, \mathbf{x}_j)$ is the pair of vertices that form the SE associated with the midpoint l.

TABLE I
DISTANCE METRICS—WHERE $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_m]^T$ AND $\mathbf{y} = [y_1, y_2, y_3, \ldots, y_m]^T$ WITH DIMENSION m

Euclidean	Manhattan	Quadratic Euclidean
$\sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$	$\sum_{i=1}^{M} \lvert\mathbf{x}_i - \mathbf{y}_i\rvert$	$\sum_{i=1}^{M}(x_i - y_i)^2$

The $\mathbf{x}$ label is given by the sort function $f(\mathbf{x})$, described in

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad H_l(\mathbf{x}) > 0 \\ -1 & \text{if} \quad H_l(\mathbf{x}) \leq 0. \end{cases} \quad (6)$$

D. Implemented Approach

The methodology implemented in this article, called NN-clas, is based on the neighbors rule. As in the *CHIP-clas*, initially, the Gabriel graph G is obtained from training set T as explained in Section II-A. The filtering stage is performed, as described by (3), and SE in the separation region is obtained.

In contrast with *CHIP-clas*, the classification performed by NN-clas is based on the distance between each test sample $\mathbf{x}$ and all support edges vertices. The label of the closest vertex is then assigned to the test sample. This approach, which simplifies the embedded implementation of *CHIP-clas* is equivalent to the application of kNN [20] with $k = 1$, however, considering only the subset of margin vertices and not all training set samples.

This approach eliminates the more costly *CHIP-clas* operations required by HME, such as the calculations of the weights of all hyperplanes, defined by (4) and the weighted sum of the hyperplanes $f(\mathbf{x})$. Also, from the memory usage perspective, only support edge vertices need to be stored.

E. Optimization Approaches

Euclidean and *Manhattan* distances are often adopted as a metric in machine learning applications. However, the former can be costly for hardware implementations[21], and the latter may impair the accuracy of the method in some applications when compared to the first one [22] and [23].

Manhattan distance (Table I) has been widely adopted in hardware implementations as a lower computational cost alternative to Euclidean distance [24]–[26] and [21]. However, it has been observed that it may impair the accuracy of the method in some applications when compared to the use of the Euclidean [22], [23], and [27]. In this article, we aimed at a distance measure that would have lower computational cost than the Euclidean distance and higher accuracy than the Manhattan.

In this context, the quadratic Euclidean distance, despite not being considered a metric for not satisfying the triangular inequality, has been often applied in problems where distance is used only ranking the samples, as are the cases of *kNN* and NN-clas. Its use does not reduce the accuracy when compared with the original Euclidean metric and still eliminates the square root operation. However, it is a bit more costly than the *Manhattan* metric, since it requires an over multiplication operation (see Table I).

Regardless of the metric adopted, to construct the Gabriel Graph, a distance matrix is needed. Since it is symmetric, only

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

the distances at the right of the main diagonal are calculated in the present implementation, considering it as an upper triangular matrix [see (7)]

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} & \dots & D_{1n} \\ D_{21} & D_{22} & D_{23} & \dots & D_{2n} \\ D_{31} & D_{32} & D_{33} & \dots & D_{3n} \\ D_{41} & D_{42} & D_{43} & \dots & D_{4n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{n1} & D_{n2} & D_{n3} & \dots & D_{nn} \end{bmatrix}. \tag{7}$$

III. Proposed Embedded System Architecture

The proposed hardware architecture co-design involves a design space exploration trade-off of a Gabriel Graph Classifier algorithm for hardware implementation to generate an RTL design of a highly-optimized FPGA accelerator onto a System-on-a-Chip (SoC) based platform. The dual-core system is configured to deal with the whole process divided into training, and classification phases. One core is running the classification, whereas the other is devoted to the time-critical training phase. As a necessity imposed by the design requirements, the classification step must run continuously. Thus its core is entirely allocated to the nonstop classifier process.

Also, the Vivado Xilinx EDA tool was used to develop the embedded system, using VHDL as a hardware description language (HDL) to implement the accelerator, and to deal with synthesis, hardware/software co-verification steps and performance, precision, cost and power design estimation parameters.

This section presents the general architecture of the developed system, according to the following subsections: (A) Co-processor Architecture, (B) Embedded System, and (C) Embedded System with Co-processor Architecture.

A. Co-Processor Architecture

The accelerator architecture was developed to speed up the Quadratic Euclidean Distance (QED) operation, hereafter named as QED architecture, and is based on a linear systolic array of processing elements (PEs) in a pipelined fashion, only requiring as many PEs as features (dimensions) and samples (elements) necessary in the system. As inputs, the architecture admits single floating-point data precision, giving as outputs the same precision. Beforehand, specific hardware blocks existing in FPGA, such as memory blocks, DSPs, logic, and interconnection types, were considered for an effective co-design process taking advantage of the intrinsic parallelism of the device. The architecture was designed using a parameterized hardware description in order to address changes in the values of the samples to be processed. The operation of each PE is associated with its own control unit (CU), thus allowing to issue different internal control signals designed in the PE, and to direct data to the different distributed local memory elements.

The proposed QED architecture, shown in Fig. 1, is a small and generic sample of the architecture for a 4 features and sample

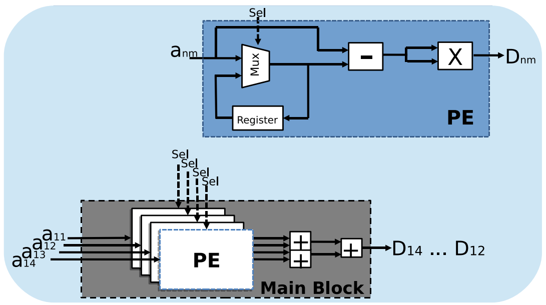


Fig. 1. Main block of the QED architecture.

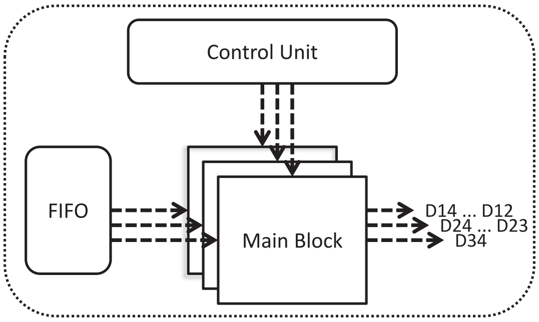


Fig. 2. QED architecture for a set of training samples.

inputs that can be extended in the number of PEs as long as hardware resources are available.

For the proposed QED architecture, only the calculation of the elements to the right of the diagonal of the matrix was considered, since the matrix is symmetric. Fig. 1 exemplifies the main block and Fig. 2 shows the whole architecture for QED operation for a set of training samples that have 1 element and 4 dimensions.

The FIFO block represents a local memory element that stores all training samples, and at each clock cycle, it provides a sample with their respective feature. Since the system uses a 32-bits floating point representation, the size of this memory element is $32 \times m \times n$, where n represents the number of training samples and m the dimension.

As discussed earlier, Fig. 2 shows the distance calculation of a set of samples of 4 elements, so the architectural design has three stages, and each stage will perform the distance calculation between one sample and the other. That is, the first stage will calculate the distance between sample a_1 and a_2, a_1 and a_3, and a_1 and a_4, the second stage will in turn calculate the distance between sample a_2 and a_3, and a_2 and a_4, already the third stage will calculate the distance between sample a_3 and a_4.

It is necessary to use multiplexers, and registers to maintain the value of a given sample, and thus perform distance calculation of this sample with the others. For example, in the first cycle of *clock* sample a_1 is supplied by the block *FIFO*, the multiplexer and delay blocks will maintain the value of this sample so that in the other cycles of *clock* receive the following samples, the distance between sample a_1 and the others can be calculated. In this manner, each multiplexer/delayer block has a selector

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

TABLE II
PROFILING OF THE EXECUTION TRAINING PHASE RUNNING ON ONE CORE OF THE ZEDBOARD ZYNQ PROCESSOR PLATFORM USING THE TARGET COMMUNICATIONS FRAMEWORK (TCF)

	System entirely in software
Distance Calculation and Proximity Graph	76,4%
Reading SD card training data	18.5%
Other Processes	5.4%

control input to indicate when the value contained in the 1 entry will be held through the delay. As long as the selector input remains at 1, the value held through the delay will be passed to the output of the multiplexer, and when the selector input is at 0 the value contained at input 1 of the multiplexer will be passed to the output of that multiplexer.

The number of multiplexers and registers present in the distance calculation architecture will depend on the number of dimensions m and the number of training samples n, as can be seen in Fig. 2 each stage has a number m of multiplexers and delays to maintain the sample characteristic values a_i, where i represents the stage number. In this way, the number of these blocks can be generically expressed according to

$$\text{num}_{\text{mult\_delays}} = m * (n-1). \tag{8}$$

The number of subtractor blocks and adders can also be expressed by (9) and (10), respectively, and the number of multiplier blocks is defined by

$$\text{num}_{\text{subtractors}} = m * (n-1) \tag{9}$$

$$\text{num}_{\text{adders}} = (n-1) * \left(\sum_{i=1}^{\log_2 m} \frac{m}{2^i} \right) \tag{10}$$

$$\text{num}_{\text{multipliers}} = m * (n-1). \tag{11}$$

The latency of the distance calculation measured in *clocks* cycles using the *quadratic Euclidean metric* is exemplified by

$$\text{Time(clks)\_Euclidian}Q = 3 + (\log_2 m) * 3 + 2. \tag{12}$$

There are two main limiting factors in this QED architecture, the number of features (dimension) and the number of samples (elements). These factors bound the implementation of different sizes of the systems, due to the number of DSPs and Memory Block allowed in the selected device, compromising the growth of resources used in a non-linear way.

The acceleration of this phase is justified based on the time complexity that is $O(n^2 m)$ (where n = number of features/dimensions and m = number of samples), as well as the presented percentage of the computational cost of the execution of the training algorithm phase using the Target Communications Framework (TCF) tool, taking samples of the program counter while running on ZedBoard Zynq processor platform (see Table II), where the processes of reading SD card of the training data along with the proximity graph and distance calculation are the most time-consuming steps within the process. For that reason, it was first decided to approach a solution for the Euclidean distance calculation due to the degree of data dependency.

B. Embedded System

The architecture of Zynq device, shown in Fig. 3, implements an Application Processor Unit (APU) into the Processing System (PS) side of the SoC and performs the whole control task of the hardware as well as the non-accelerated steps of training phases in software in one core, and the classification phases in the other. The data inputs are stored in a SD Card that is also controlled by PS.

The data bus is connected to the APU Zynq by means of the port setup to 64 bits to ensure reliable communication between the PS and the *Programmable Logic* (PL). The general purpose port of the Zynq PS (*GP_AXI*) is used to configure the rest of the peripherals registers with the *AXI Lite*.

The proposed architecture is designed to obtain a high flow rate in the data transfer to the *hardware* accelerator. The transfer control is executed in the IP AXI DMA, which handles the data movement from readings and writings mapped in memory in its port *AXI_MM* for the connection with the co-processor in port *AXI_STREAM*.

In this way, the communication interface aims at minimizing the cost of memory transfers to the CPU by leaving the DMA drive in charge of moving the data with greater flow in the protocol AXI Stream.

C. Embedded System With Co-Processor Architecture

The block diagram of the implemented co-processor is represented in Fig. 3. In this architecture, the core used to compute quadratic Euclidean distances has its inputs provided by the controller block, and its outputs are stored in a dedicated BRAM unit for distance values. As shown in the diagram, one of the advantages of the hardware implementation is the use of 1024-bit-wide custom buses for transferring all 32 dimensions of a sample in a single clock cycle in a burst mode.

The controller unit, in turn, receives the data from the AXI_STREAM port and uses another BRAM block to store the samples. Synchronization for receiving data from DMA transfers uses a simple producer/consumer algorithm to supply samples to the QED architecture and then to send the calculated distances back to the processor. In addition, the QED architecture receive a sequence of samples and generate at the outputs a sequence with the quadratic Euclidean distances between those samples.

To obtain the complete set with all pairwise distances, the consumer process must traverse the data set N times, using a variable to store the number of the current iteration. With this procedure, it is avoided that the distance between two samples is recalculated repeatedly.

Internally, the QED architecture data path uses a structure in pipeline to compute integer blocks with M dimensions of a pair of samples in a parallel way. With the use of M subtractors and multipliers in parallel followed by a tree of adders to accumulate the sum in the M dimensions, the co-processor can reduce the originally linear cost in the number of cycles for computing the quadratic Euclidean distance to a logarithmic cost. In addition, maintaining the pipeline in the data path in constant operation

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

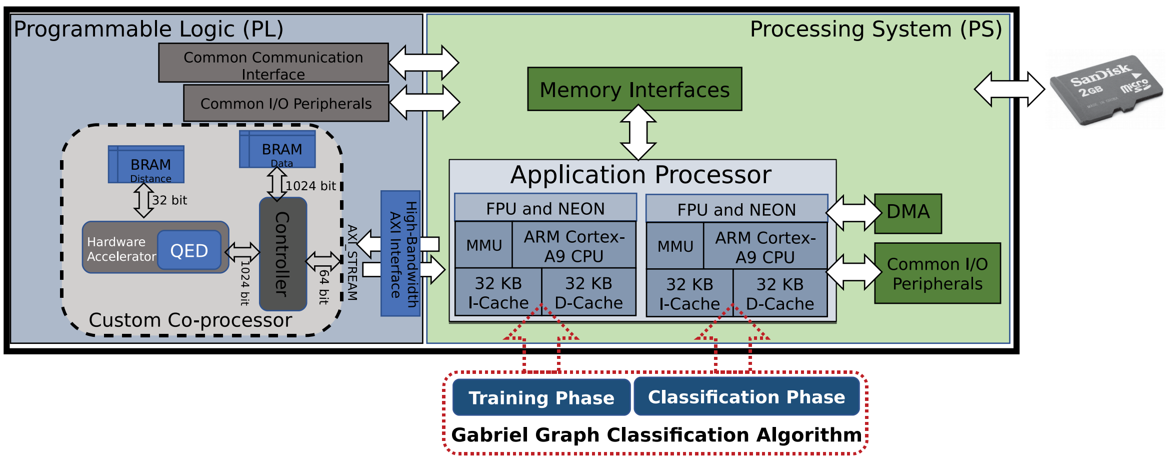


Fig. 3. Overall internal structure design of the system with accelerators.

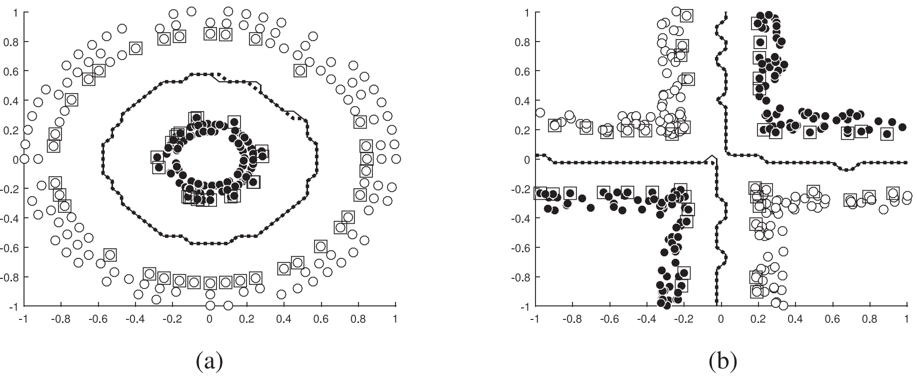


Fig. 4. Decision surfaces generated by the NN-clas (dotted lines) method and the CHIP-clas (continuous lines). (a) Benchmark cluster. (b) Benchmark corners.

reduces the time required to process a block of values in the distance matrix.

The developed architecture should provide, in addition to the gains with parallelism and pipeline, the added advantage of the implementation in dedicated hardware. Since it is a single purpose solution described directly at the transfer level between registers, execution in the co-processor does not depend on search, decode and instruction processing cycles, which allows a much greater efficiency in the execution of the two external loops for calculating the distance between samples.

It is expected, therefore, that the execution of the accelerated system with the developed hardware will be faster than the execution of the system entirely in software. As long as the DMA transfer-based interface and the synchronization mechanism implemented with producer/consumer algorithms on the AXI Stream protocol can provide bandwidth and flow compatible with the expected data stream for the system, the implementation of the co-processor should allow for performance gains.

IV. Results

The first experiments were designed to validate the proposed NN-clas and to evaluate its performance. Synthetic 2-D data was generated first in order to obtain training data and results that could be visualized. After that, the implemented system was applied to real data.

In the first experiments the decision surfaces obtained by the proposed methodology and CHIP-clas, could be compared in the figures that follow. In Fig. 4 the square points represent the vertices of the support edges. The dotted and continuous lines represent, respectively, the separation surfaces obtained by NN-clas and CHIP-class. It is clear that both separation surfaces are similar, and, in some cases, the difference is almost unnoticeable, so NN-clas also tends to generate surfaces equidistant from the two classes, thus maximizing the separation margin.

The remaining experiments were carried out on 13 real data sets taken from the UCI Machine Learning Repository, and also on 2 gene-expression problems: Golub [28] and “BcrHess” [29]. All samples with missing values were removed, and features were normalized between $\{-1, 1\}$. A cross-validation technique was used, aiming to guarantee the statistical relevance. As a measure of performance, the AUC (area under the ROC curve) was chosen, since some of the data sets are unbalanced.

In this work, the following methods were compared: *kNN* with $k = 1$ (NN), the method proposed here (NN-clas), CHIP-clas, and reduced CHIP-Clas hereafter referred to as **RCHIP-clas**. *kNN* was chosen since it is quite similar to the NN-clas. However, it is important to notice that *kNN* calculates the distance between

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

TABLE III
RESULTS OF ALGORITHMS

Data base	CHIP-clas	NN-clas	RCHIP-clas	NN	N	N_d	N_t	N_b
Australian Cr.	0.846 ± 0.041	$\mathbf{0.848 \pm 0.041}$	0.846 ± 0.04	0.790 ± 0.047	690	14	395	62
Banknote Auth.	0.980 ± 0.026	$\mathbf{0.999 \pm 0.003}$	0.979 ± 0.028	0.998 ± 0.003	1372	4	1235	61
BcrHess	0.811 ± 0.117	$\mathbf{0.814 \pm 0.125}$	0.807 ± 0.123	0.714 ± 0.228	133	30	71	6
B. Cancer W.P	0.956 ± 0.029	$\mathbf{0.964 \pm 0.028}$	0.963 ± 0.026	0.949 ± 0.031	683	9	520	20
Climate M.S.C.	$\mathbf{0.840 \pm 0.068}$	0.771 ± 0.129	0.768 ± 0.131	0.594 ± 0.088	540	18	274	199
Fertility	$\mathbf{0.588 \pm 0.259}$	0.553 ± 0.162	0.559 ± 0.163	0.578 ± 0.224	100	9	48	9
German Cr.	0.672 ± 0.042	$\mathbf{0.685 \pm 0.033}$	0.668 ± 0.043	0.605 ± 0.064	1000	24	463	263
Golub	0.774 ± 0.172	0.788 ± 0.160	0.784 ± 0.160	$\mathbf{0.789 \pm 0.086}$	72	50	39	9
Haberman's S.	0.569 ± 0.089	0.559 ± 0.108	$\mathbf{0.579 \pm 0.084}$	0.539 ± 0.082	306	3	143	49
ILPD	0.560 ± 0.086	0.568 ± 0.087	0.559 ± 0.090	$\mathbf{0.595 \pm 0.072}$	579	10	275	117
Liver disorders	$\mathbf{0.614 \pm 0.102}$	0.598 ± 0.114	0.566 ± 0.124	0.597 ± 0.079	345	6	164	110
P. ind. diabetes	0.721 ± 0.036	$\mathbf{0.728 \pm 0.046}$	0.716 ± 0.049	0.672 ± 0.058	768	8	392	124
Parkinsons	0.898 ± 0.149	0.894 ± 0.146	0.895 ± 0.147	$\mathbf{0.945 \pm 0.125}$	195	22	175	62
Sonar. M against R.	$\mathbf{0.876 \pm 0.079}$	0.872 ± 0.061	0.846 ± 0.067	0.872 ± 0.061	208	60	187	163
Statlog heart	$\mathbf{0.798 \pm 0.082}$	0.788 ± 0.071	0.784 ± 0.160	0.789 ± 0.086	270	13	134	36
Average rank $R(\mathcal{L})$	**2.0333**	**2.0333**	**3.1000**	**2.8333**				

each test sample $\mathbf{x}$ and the entire training set. This requires the storage of all training samples instead of just storing the vertices of the support edges as in the NN-clas. Furthermore, since *kNN* has no noise filtering phase, the classification of overlapping data can be affected, leading to overfitting. One way to minimize *overfitting* would be through an exhaustive search for the best k. However, this solution is not feasible for embedded systems, demonstrating that the NN-class method is much more suitable to be embedded.

Table III shows the mean AUC and the standard deviation obtained by all methods in each data set. The total number of samples N, the dimensionality N_d, the number of training samples N_t, and the total number of vertices of supporting edges N_b are also presented in the table. It is possible to notice the reduction of data storage required by the proposed method about *kNN*, as well as the complexity of each data set. The values highlighted in bold indicate the best performance on each data set.

In order to statistically compare more than two method results Friedman's test was applied, which is based on a ranking of methods for all data sets. The last line of Table III shows the mean *rank* obtained by each of the classifiers. It is important to note that the lower the value of *rank*, the better is the performance of the algorithm, assuming as null hypothesis H_0 the equivalence between the four methods with a significance level of $\alpha = 0.05$. The value p obtained with this test was 0.04, which means that it is not possible to state that the methods are statistically equivalent. After this statistical test, a matched *post-hoc* test (Wilcoxon) was performed to verify statistical significance between each pair of algorithms. For all tests, the null hypothesis H_0 was the equivalence between the methods compared to a significance level of $\alpha = 0.05$. The $p - value$ obtained in each paired test are NN-clas/CHIP-clas (0.476) and RCHIP-clas/CHIP-clas (0.036). It can be observed that NN-clas and CHIP-clas are statistically equivalent, and RCHIP-clas has lower performance compared to CHIP-clas.

In order to implement the NN-class, the accelerated co-processor was implemented in *VHDL*, using the System Generator to validate the functionality and to explore the user guide interface of the Matlab Simulink to facilitate the comparisons of the results. The code of all co-processor PEs was generated by a MATLAB program, with user-defined parameters to change any

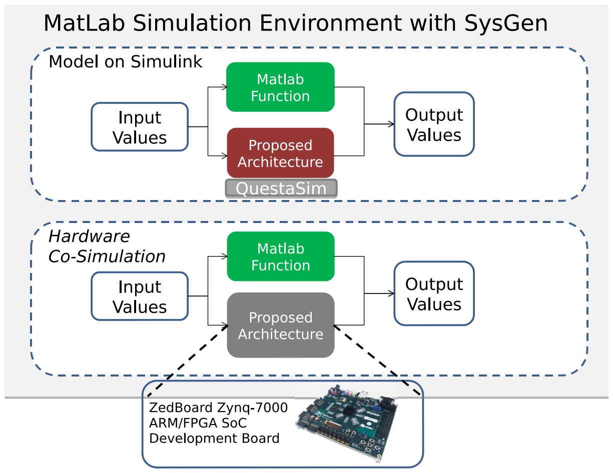


Fig. 5. Functional and hardware co-simulation of the proposed QED architecture.

general configuration of the architecture and then, a functional simulation was performed by *QuestaSim* simulator (see Fig. 5).

Data was read directly from a text file. However, the same structure of data transfer defined in the architecture was used, thus facilitating the simulation of the project.

In order to validate the PEs for distance calculation, random samples were generated through the Matlab *rand* function in the range [0 100]. The generated data were converted into binary according to the simple precision texting IEEE-754 standard.The results provided by the PE were stored in another text file by the control unit. After simulation, the stored results were read and converted to decimal format by the developed Matlab program and then compared with the values obtained by the same algorithm implemented only in software.

The results were compared using mean squared error (MSE) and are presented in Table IV. The first column shows the number of samples and its size, and, the second and third the results of MSE.

It can be observed from the table that the error increases as the number of dimensions grows since with a larger number of dimensions, more sum operations are performed. Furthermore, a Hardware Co-Simulation step was then performed to analyze the QED architecture mapped onto the ZedBoard Zynq-7000

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
DISTANCE CALCULATION ERROR

Data Size	MSE *Manhattan*	MSE Euclidean quadratic
$4x4$	$1.8853e^{-11}$	$6.1767e^{-07}$
$16x4$	$4.5838e^{-11}$	$5.6288e^{-07}$
$30x8$	$2.29373e^{-10}$	$2.4916e^{-06}$
$60x10$	$7.6504e^{-10}$	$5.7339e^{-06}$
$120x20$	$5.5677e^{-09}$	$3.6662e^{-05}$

ARM/FPGA SoC Development Board in order to explore different data size (see Fig. 5).

A. Embedded System Architecture Results

Co-processor results are compared to the performance of a classifier implemented in *software*, in a bare-metal scheme, on an ARM processor available in the used Zynq SoC device. Both are applied to a synthetic classification problem composed of 1000 samples. Other relevant metrics for the validation of the system are described, such as the behavior of the classifier as the problem scales up and power analysis and resources consumption.

The architecture was implemented in the *Zynq* device available in the *ZedBoard* kit, and as mentioned earlier, the system used a single precision floating-point representation of the data. The improvement in execution times is measured by counters embedded in the *Zynq APU* in three execution modes to obtain a comparative analysis between the performance of different computational methods for calculation of the quadratic Euclidean distance:

1) **FPGA co-processor system:** The distance matrix is computed in the co-processor described in Section III-A;
2) **System entirely in *software*:** The matrix of distances is calculated by the C language implementation for calculating the distance between samples;
3) **Co-processor system for vector arithmetic operations:** The distance matrix is calculated in the same algorithm implemented in C language, but with compilation optimizations enabled for the NEON co-processor instruction set.

The third method is available on the *Zynq* platform through the NEON co-processor embedded in the *SoC APU*. This component is a hardware accelerator built into the *Zynq* CPU peripheral set, and provides an extension to the ARM instruction set for vector and floating-point operations [30].

As this unit implements a set of SIMD (Single Instruction Multiple Data) instructions to execute vector computations, its operations are inferred by the C compiler to obtain higher performance in code segments for logical and arithmetic operations in blocks of data. The NEON unit, therefore, is an accelerator commonly used in signal processing and computer graphics applications and should offer a performance gain for the calculation of the quadratic Euclidean distance comparable to that of the co-processor developed in this work.

For running the tests and assessing the execution time of the methods listed, the following software interfaces are used, which

TABLE V
MEASURED RUNTIME FOR EACH SYSTEM VERSION DURING TESTS

	Distance calculation runtime (ms)
System entirely in software	1336.7
NEON co-processor system	235.2
System with co-processor in FPGA	166.5

receive a pointer to the data matrix `data` as a parameter and write the array of distances in the memory block pointed to `dist`.

```
  void
  CalDistHW(const float **data, float **
dist);
  void __attribute__((optimize("-O0")))
  CalDist(const float **data, float **dist);
  void
  CalDistNEON(const float **data, float **
dist);
```

In the `CalDist` function, used to test the system entirely in *software*, the attribute `optimize` is set to `O0` in the `__attribute__` directive included in the function statement. This attribute configures the compiler not to perform optimizations on the code in this routine. In the interface defined in `CalDistNEON`, in turn, the optimization level follows the value `O2` specified in the system configuration. Xilinx SDK v2017.4 IDE disassembly tool shows the following assembly code generated for this procedure after the optimizations:

```
vldmia   r3!, {s15}
vldmia   r2!, {s13}
cmp  r3,  r0
vsub.f32      s15,  s15,  s13
vmla.f32      s14,  s15,  s15
bne  21c <CalDistNEON+0x48>
vstmia   r1!, {s14}
```

Therefore, the function for calculating the distance matrix in the NEON co-processor includes instructions from that module. In the *assembly* code snippet, the `vldmia` and `vstmia` are mnemonic for Vector Load Multiple Incremental Address and Vector Store Multiple Incremental Address, respectively, and the instructions `vsub.f32` and `vmla.f32` denote the operations *Vector Subtract 32 bits float* and *Vector Multiply Accumulate 32 bits float* [31]. The gain of performance in the optimized code results, therefore, in the greater efficiency of block memory access to the vectors and the execution of the multiplication and summation operations condensed in the same statement `vmla.f32`.

The measurement of the performance of the co-processor in the developed FPGA shows an acceleration factor of $8x$ about the execution in *software*, as can be seen in Table V. The system also presents a performance comparable to the execution of the classification with the accelerator NEON, whose acceleration in relation to the *software* reaches 5.7x.

If compared to the NEON co-processor, the FPGA solution has a slightly shorter runtime. The ratio between the

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

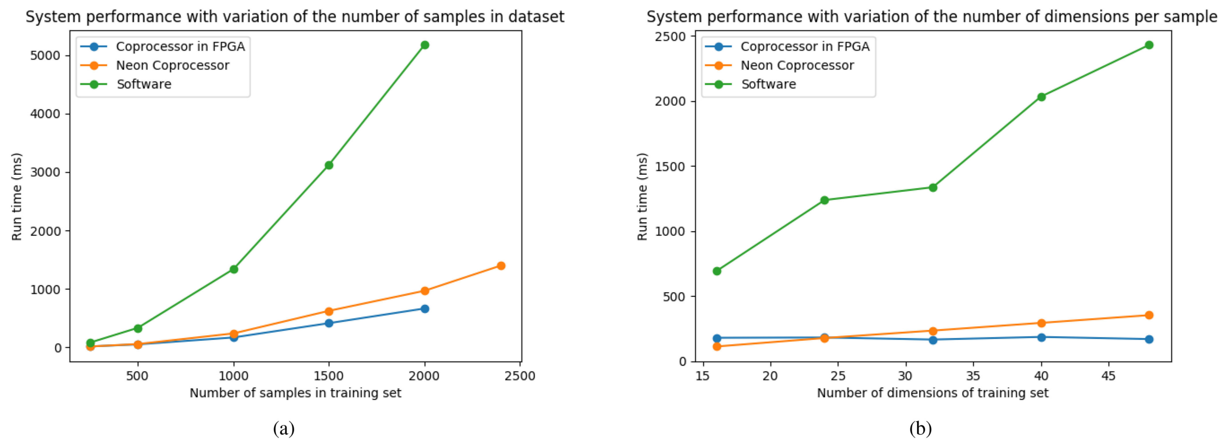
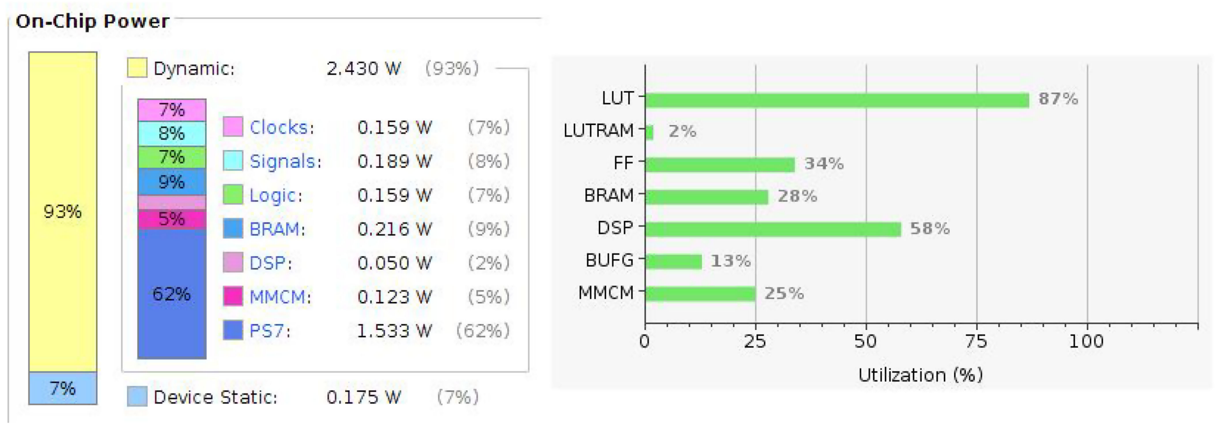


Fig. 6. Execution times measured in distance matrix. (a) Computing in 32-dimensional problems. (b) Computing in problems with 1000 samples.


On-Chip Power

Dynamic:	2.430 W	(93%)
Clocks:	0.159 W	(7%)
Signals:	0.189 W	(8%)
Logic:	0.159 W	(7%)
BRAM:	0.216 W	(9%)
DSP:	0.050 W	(2%)
MMCM:	0.123 W	(5%)
PS7:	1.533 W	(62%)
Device Static:	0.175 W	(7%)

LUT	87%
LUTRAM	2%
FF	34%
BRAM	28%
DSP	58%
BUFG	13%
MMCM	25%

Fig. 7. Power analysis and resources consumption.

results in these two methods shows a 29.2% reduction in the FPGA, which, although limited to an absolute value of 68.7 ms, may be significant for high-throughput applications in *hardware*.

Another important metric for studying the performance of the developed system is how it scales-up with volume and dimensionality. For this, the classifier runs for data sets of a different number of samples and number of dimensions. Again its performance is analyzed in comparison to that of the classifier with the NEON accelerator and with the system implemented entirely in *software*. Because the FPGA co-processor has been implemented as a single-application hardware unit, its configuration to handle different size problems requires new compilation and routing of the parameterized source code in VHDL. This system, therefore, does not have the flexibility to change its dimension at running time, and can only change with new RTL synthesis of the developed circuit.

The results of the measured execution times for the computation of the quadratic Euclidean distance in the different problems are shown in Fig. 6(a). The maximum system capacity remains limited to 2400 samples or 48 dimensions, values beyond which the Zynq RAM or the availability of FPGA resources are exceeded.

We are keeping the number of samples fixed at 1000, and by varying the number of dimensions treated in the classification problem, the pattern shown in the graph of the Fig. 6(b) is verified for the execution times in the calculation of the matrix of distances.

It can be seen in the graph of Fig. 6(a) that all modes of execution of the system obtain the quadratic complexity pattern with the number of samples.

The FPGA co-processor implementation achieves better performance across all measurements, remaining marginally superior to the NEON co-processor for up to 1000 samples and following a lower growth trend for larger scale problems. The version of the system entirely in *software*, in turn, shows a higher rate of increase in execution times for each increment of the number of samples tested.

Fig. 6(b) shows the linear pattern of complexity for computing the distance matrix. Similarly to the behavior in the graph of Fig. 6(a), it is observed that the three versions of the system obtain different growth rates as the dimension scales up. Again, the solution in *software* keeps a running time increment higher than the other implementations for each test. Also, the FPGA implementation shows a practically constant pattern in time measurement for all considered dimension numbers.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

These results point out to the little dependence of the implemented co-processor with input space dimension since the factors that determine the performance of the system in FPGA are the traffic volume in DMA transactions and the synchronization algorithm used in the interface with the CPU. This behavior also leads to the point of intersection between the curves of the FPGA system and NEON in Fig. 6(b). The graph shows that, for problems with less than 24 dimensions, NEON performance outperforms that of the dedicated co-processor, which indicates that the developed system is advantageous to handle problems with a larger number of dimensions. At the same time, it is affected by the effect of *overhead* communication required for the FPGA for small-scale problems.

Finally, it is possible to view the power analysis and resource consumption presented in Fig. 7, where it is worth noting that the whole system uses 2.43 W for the dynamic consumption and there are still resources to implement other accelerators and allow a higher degree of parallelism at other stages of the training process, as reported by Vivado tool.

V. Conclusion

As shown in this article, a large margin classifier was embedded in a system as a SoC and had its performance enhanced by integrating a dedicated hardware module. As proposed, the implementation could be accomplished appropriately on a Zynq development platform using the available FPGA resources and without exceeding the memory limitations of the SoC. As was seen by the results of tests, despite the peculiarities used in the design of this system with reconfigurable hardware, the proposed system was feasible to embed an application of computational intelligence. The study and implementation developed in this article show a simple and effective architecture to accelerate the execution of an algorithm in a platform that integrates embedded software and dedicated hardware.

As can be observed by results, there was a gain when integrating a dedicated hardware module into the system as a co-processor in FPGA. The project obtained a throughput up to 8.9 MB/s in the write operations and an acceleration factor of $8x$ concerning the execution of the algorithm in software, without taking into account compilation optimizations.

Finally, the two solutions tested for the acceleration of the algorithm, based on the instructions set extended to NEON and implemented in FPGA, present, therefore, complementary aspects for its use in the proposed type of application. The former has a *de facto* development cost much lower than the latter, since it starts from the implementation in *software* with the inference of vector operations in the compilation. However, its linear cost with the number of samples makes its performance worse than the FPGA solution for large problems.

Acknowledgment

The authors would like to thank *Pró-Reitoria de Pesquisa (PRPq)* of the *Universidade Federal de Minas Gerais* (UFMG), FAPEMIG, CNPq, and CAPES for financial support. The authors would also like to thank the Xilinx University Program (XUP) for the software licenses provided.

References

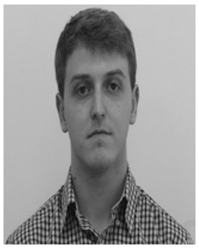
[1] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Commun. Surv. Tut.*, vol. 20, no. 1, pp. 416–464, Jan.–Mar. 2018.
[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[3] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.
[4] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[5] A. H. Sodhro, S. Pirbhulal, and V. H. C. de Albuquerque, "Artificial intelligence-driven mechanism for edge computing-based industrial applications," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4235–4243, Jul. 2019.
[6] W. Shi and S. Dustdar, "The promise of edge computing," *IEEE Comput.*, vol. 49, no. 5, pp. 78–81, May 2016.
[7] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine learning for predictive maintenance: A multiple classifier approach," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 812–820, Jun. 2015.
[8] G. Constantinides and N. Nicolici, "Guest editors' introduction: Surveying the landscape of FPGA accelerator research," *IEEE Des. Test Comput.*, vol. 28, no. 4, pp. 6/7, Jul.–Aug. 2011.
[9] M. Papadonikolakis and C. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1040–1052, Jul. 2012.
[10] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Work-in-progress: A power-efficient and high performance FPGA accelerator for convolutional neural networks," in *Proc. Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2017, pp. 1–2.
[11] F. Conti *et al.*, "An IoT endpoint system-on-chip for secure and energy-efficient near-sensor analytics," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 64, no. 9, pp. 2481–2494, Sep. 2017.
[12] A. Pullini, F. Conti, D. Rossi, I. Loi, M. Gautschi, and L. Benini, "A heterogeneous multi-core system-on-chip for energy efficient brain inspired computing," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 2910.
[13] F. Ortega-Zamorano, J. M. Jerez, D. U. Muñoz, R. M. Luque-Baena, and L. Franco, "Efficient implementation of the backpropagation algorithm in FPGAs and microcontrollers," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1840–1850, Sep. 2016.
[14] H. M. Hussain, K. Benkrid, and H. Seker, "An adaptive implementation of a dynamically reconfigurable k-nearest neighbour classifier on FPGA," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2012, pp. 205–212.
[15] J. Su, J. Liu, D. B. Thomas, and P. Y. Cheung, "Neural network based reinforcement learning acceleration on FPGA platforms," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 4, pp. 68–73, 2017.
[16] Y. Zhu, M. Mattina, and P. Whatmough, "Mobile machine learning hardware at ARM: A systems-on-chip (SoC) perspective," 2018, *arXiv:1801.06274*.
[17] L. Torres, C. Castro, F. Coelho, F. S. Torres, and A. Braga, "Distance-based large margin classifier suitable for integrated circuit implementation," *Electron. Lett.*, vol. 51, no. 24, pp. 1967–1969, 2015.
[18] K. R. Gabriel and R. R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Biol.*, vol. 18, no. 3, pp. 259–278, 1969.
[19] L. P. Garcia, A. C. de Carvalho, and A. C. Lorena, "Effect of label noise in the complexity of classification problems," *Neurocomputing*, vol. 160, pp. 108–119, 2015.
[20] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *Amer. Statist.*, vol. 46, no. 3, pp. 175–185, 1992.
[21] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2011, pp. 248–255.
[22] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware," in *Proc. ACM/SIGDA 9th Int. Symp. Field Programmable Gate Arrays*, 2001, pp. 103–110.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.

[23] M. L. James Theiler, M. Estlick, and J. J. Szymanski, “Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery,” *Proc. SPIE*, vol. 4132, 2000, Art. no. 100.
[24] E. S. Manolakos and I. Stamoulias, “IP-cores design for the KNN classifier,” in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 4133–4136.
[25] H. M. Hussain, K. Benkrid, and H. Seker, “An adaptive implementation of a dynamically reconfigurable k-nearest neighbour classifier on FPGA,” in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, Jun. 2012, pp. 205–212.
[26] I. Stamoulias and E. S. Manolakos, “Parallel architectures for the KNN classifier–design of soft IP cores and FPGA implementations,” *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 2, 2013, Art. no. 22.
[27] M. Estlick, “An FPGA implementation of the k-means algorithm for image processing,” Ph.D. dissertation, M.S. thesis, Dept. Elect. Eng., Northeastern Univ., Boston, MA, USA 2002.
[28] T. R. Golub *et al.*, “Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring,” *Science*, vol. 286, no. 5439, pp. 531–537, 1999.
[29] K. R. Hess *et al.*, “Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer,” *J. Clin. Oncol.*, vol. 24, no. 26, pp. 4236–4244, 2006.
[30] L. Crockett, R. Elliot, M. Enderwitz, and R. Stewart, *The Zynq Book: Embedded Processing Withe ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Glasgow, U.K.: Strathclyde Academic Media, 2014.
[31] *ARM Compiler Toolchain - Assembler Reference*, ARM, Jul. 2011.

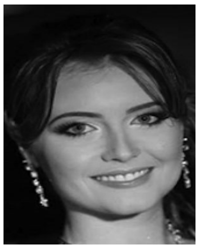
Janier Arias-Garcia received a B.Sc. degree in physics engineering from the Universidad del Cauca, Popayán, Colombia, in 2007, and the master and a Ph.D. degrees in mechatronic Systems from the Universidade de Brasília, Brasília, Brazil, in 2010 and 2014 respectively.

He is currently a Adjunct Professor in the Department of Electronic Engineering (DELT) *Universidade Federal de Minas Gerais* (UFMG) Belo Horizonte, Brazil, and a member of the MACRO group. He has experience mainly in the development of architectures and in custom hardware applications to implement numerical algorithms, involving theory and practice of using reconfigurable hardware systems (FPGAs) and their synthesis tools to describe digital circuit synthesis techniques, allowing a controlled exchange between stability, numerical precision, and metrics such as area, energy, and performance. His current research interests are in the development of digital hardware architectures for processing matrix calculations and classifiers, as well as cryptosystem architectures for embedded systems applications.



Augusto Mafra received a B.Sc. degree in electrical engineering from the Universidade Federal de Minas Gerais (UFMG) Belo Horizonte, Brazil, in 2018, with emphasis on software engineering, embedded systems and electronic design automation.

Since 2018, he has been a Software Engineer with Cadence Design Systems, Belo Horizonte, acting on compilers and formal verification.



Liliane Gade received the B.Sc. degree in eletronic engeneering from Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Brazil, in 2013, and the M.Sc. degree in electrical engeneering from Universdade Federal de Minas Gerais (UFMG), Brazil, in 2018.

She is currently is a Researcher with Empresa Brasileira de Pesquisa e Inovação Industrial- EMBRAPII, unidade DCC-UFMG, Belo Horizonte. Her current research interests include embedded systems, computational intelligence and machine learning.

Frederico Coelho received the B.Sc. and M.Sc. degrees from Universidade Federal de Minas Gerais (UFMG) Belo Horizonte, Brazil, iin 1998 and 2009, respectively, and a joint Ph.D. degree from UFMG and Université Catholique de Louvain, Louvain-la-Neuve, Belgium, all in electrical engineering. He is currently an Adjunct Professor with UFMG, where he integrates the Computational Intelligence Laboratory. His current research interests include computational intelligence, soft computing and machine learning.

Cristiano Castro received a B.Sc. degree in computer science from Universidade Federal de Lavras, Lavras, Brazil, in 2001, and the M.Sc. and Ph.D. degrees in communications and computer engineering from Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brazil, in 2004 and 2011, respectively.

Since 2014, he has been with the Electrical Engineering Department at UFMG where he has been teaching and supervising students of the undergraduate courses in electrical, systems, and control and automation engineering. His current research interest involves theoretical aspects and applications of machine learning and data mining.

Luiz Torres received the B.Sc degree in computer science in graph theory from the University Center of Belo Horizonte, Belo Horizonte, Brazil, in 2010, the master's and Ph.d degrees in electrical engineering from Universidade Federal de Minas Gerais, Belo Horizonte, in 2012 and 2016, respectively, where he also received the post-doctorate through the Brazillian agency National Council for Scientific and Technological (CNPq), in 2017. He received the postdoctoral degree in international cooperation with Optical Networks Laboratory (ONLAB), KTH Royal Institute of Technology, Stockholm, Sweden, in 2018

Since 2019, he has been with the Computer and Systems Department, Universidade Federal de Ouro Preto (UFOP), Ouro Preto, Brazil, where he is now an Adjunct Professor and Head of the Machine Learning Research Group.

Antônio Braga (Member, IEEE) received a B.Sc degree in electrical engineering and a master's degree in computer science from Universidade Federal de Minas Gerais (UFMG) Belo Horizonte, Brazil, in 1987 and 1991, respectively, and the Ph.D. degree in electrical engineering in recurrent neural networks from from, Imperial College, University of London, London, U.K., in 1995

Since 1991, he has been with the Electronics Engineering Department at the UFMG where he is now a Full Professor and Head of the Computational Intelligence Laboratory. He is also an Associate Researcher of the Brazilian National Research Council. As a Professor and Researcher he has co-authored many books, book-chapters, and journal and conference papers. He is an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS and *Neural Processing Letters*.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE MINAS GERAIS. Downloaded on May 11,2022 at 12:13:14 UTC from IEEE Xplore. Restrictions apply.